MASTER EN SCIENCES INFORMATIQUES

Analyse et Développement d'un Prototype X.435 Utilisant X.400 Version 1984

Kao, Wei-Chao

Award date:
1992

Awarding institution:
Universite de Namur

Facultés Universitaires Notre-Dame de la Paix
Institut d'Informatique

Analyse et Développement d'un Prototype X.435 Utilisant X.400 Version 1984

par KAO Wei-Chao

Mémoire présenté en vue de l'obtention
du titre de Licencié et Maître en Informatique

Promoteur : Professeur Ph. Van Bastelaer

Année académique 1991-1992

Rue Grandgagnage, 21 - 5000 Namur (Belgium)

Nous remercions tout d'abord notre promoteur Professeur Philippe Van Bastelaer, pour ses précieux conseils tout au long de ce travail.

Nous remercions également Monsieur Patrick Geurts de la société BIM, qui nous a permis d'effectuer un stage dans les meilleures conditions.

De nombreux membres de BIM nous ont aidés durant le stage. Nous tenons ainsi à remercier Stefano Preverdi, Leen Gossens, Philippe Cayphas, Moreno Di Mattia, et tous ceux que nous aurions oubliés.

Finalement, nous adressons nos plus vifs remerciements à Monsieur Olivier Dubois, membre de BIM, qui nous a consacré beaucoup de son temps précieux. Nous lui témoignons de la plus sincère reconnaissance pour son aide et ses conseils dans ce travail.

Résumé

Ce travail analyse comment la Recommandation X.400 version 1984 de CCITT peut être utilisée pour réaliser la Recommandation X.435 de CCITT. Nous donnons d'abord une brève introduction à l'EDI (Echange de Données Informatisé), à X.400 et à X.435, dans laquelle nous insistons sur le codage EDIFACT et sur les différences entre les deux versions de X.400. Ensuite, nous tentons de mettre en évidence les problèmes que l'on pourrait rencontrer dus à l'utilisation de X.400 version 1984 à la place de X.400 version 1988 pour X.435, notamment ceux concernant la sécurité. Finalement, nous donnons une description du développement du prototype X.435 que nous avons réalisé en utilisant X.400 version 1984.

Abstract

This work analyses how the CCITT X.400 Recommendation version 1984 can be used to carry out the CCITT X.435 Recommendation. We give first a brief introduction to EDI (Electronic Data Interchange), to X.400 and to X.435, in which we lay stress on the EDIFACT coding and on the differences between the two versions of X.400. Next, we attempt to underscore the problems that we may encounter due to the use of X.400 version 1984 instead of X.400 version 1988 for X.435, notably those relating to the security. Finally, we give a description of the development of the X.435 prototype that we've carried out by using X.400 version 1984.

Table des Matières

Annexe A : Liste des Erreurs du PDU Pedi

Annexe B : PDU Pedi (version corrigée)

Annexe C : PDU Pedi (version "structurée")

Introduction

L'EDI est une grande révolution dans le monde économique et le monde informatique à cette fin du vingtième siècle. Cette technique permet des gains de temps et de productivité appréciables, et supprime les barrières linguistiques et commerciales qui existent aujourd'hui entre les partenaires commerciaux. Tout le monde s'accorde à prévoir à l'EDI un brillant avenir.

Or, nous constatons que l'EDI a encore beaucoup de mal à prendre son envol. Cette situation est principalement due aux nombreux standards de communication et de représentation utilisés, qui empêchent des partenaires commerciaux utilisant des standards différents d'entrer en communication pour s'échanger des documents EDI. Pour que l'EDI puisse se généraliser avec succès, il faut impérativement trouver un standard de communication et un standard de représentation universels afin de rendre le monde EDI actuel plus ordonné et plus homogène. C'est chose faite aujourd'hui, avec la désignation de X.400 - le service de messagerie électronique défini par CCITT - comme le standard de communication universel. Quant au standard de représentation, on préconise l'utilisation d'EDIFACT, le codage défini par les Nations-Unies. En outre, CCITT a défini la Recommandation X.435, le service de messagerie EDI qui régit l'utilisation de X.400 version 1988 pour transférer des documents EDI.

Ce texte est consacré à l'utilisation de X.400 pour l'échange EDI, et plus particulièrement à l'utilisation de X.400 version 1984 pour X.435. Le **premier chapitre** donne une introduction technique à l'EDI. Nous allons définir avec précision le terme "EDI", présenter les standards universels choisis - X.400 et EDIFACT - et la raison de ce choix. Nous verrons aussi comment les partenaires commerciaux peuvent aujourd'hui contourner le problème de l'incompatibilité des standards en faisant appel aux services des VANs. Ensuite, nous discuterons du codage EDIFACT, en présentant la structure générale d'un interchange EDIFACT et ses différents composants: code, élément de donnée simple, élément de donnée composite, segment et message. Finalement, nous parlerons du message EDIFACT, en voyant comment un message est défini par un diagramme de branchement ou par une table de segments. Les règles de codage seront également abordées.

Le **deuxième chapitre** est consacré à la Recommandation X.400. La présentation de X.400 sera divisée en deux parties: les fonctionnalités communes des deux versions de X.400, c'est-à-dire les versions 1984 et 1988, puis les fonctionnalités qui sont propres à la version 1988. Nous présenterons l'architecture générale de X.400, ainsi que la structure des messages utilisés dans le MTS X.400. Le système de messagerie interpersonnelle, la seule application initialement prévue dans X.400, sera également abordé. Le problème d'adressage et la notion d'*O/R Name* seront ensuite discutés. Enfin, nous parlerons des fonctionnalités additionnelles de X.400 version 1988: le *Message Store*, le PDAU, l'utilisation des services de répertoire X.500, et les services de sécurité.

Le **troisième chapitre** est consacré à la Recommandation X.435, le service de messagerie EDI défini par CCITT. X.435 spécifie la manière dont X.400 version 1988 doit être utilisé pour l'EDI et représente le point de jonction entre le standard de communication et le standard de représentation. Après la présentation de l'architecture générale de X.435, nous parlerons

longuement de la structure du message EDI (EDIN) et de la notification EDI (EDIN). Ensuite, nous donnerons une introduction à l'importante notion de responsabilité EDI, ainsi qu'au relais de message EDI et au renvoi de notification EDI.

Le **quatrième chapitre** analyse la faisabilité de l'utilisation de X.400 version 1984 pour X.435. En effet, X.435 est basé sur X.400 version 1988, mais ce dernier est une Recommandation récente et encore très peu utilisée. Nous essaierons donc de voir si l'on peut contourner ce problème en utilisant X.400 version 1984 à la place de X.400 version 1988 pour réaliser X.435, et de trouver les problèmes que l'on pourrait rencontrer à cause de ce changement. Nous discuterons de l'absence du *Message Store*, de la difficulté d'intégration de X.500 dans X.400 version 1984, des services de sécurité manquants, et de l'utilisation du MTA version 1984 pour X.435. Nous insisterons tout particulièrement sur les services de sécurité, dont l'absence est à notre avis le problème principal de l'utilisation de X.400 version 1984 pour X.435. Pour terminer, nous donnerons la liste complète des éléments de service qui manquent à X.400 version 1984 pour réaliser X.435.

Finalement, le **cinquième chapitre** traite du développement du prototype X.435 que nous avons créé en utilisant X.400 version 1984. Nous parlerons d'abord des outils utilisés pour le prototype, notamment le *X.400 Gateway API*, le *X.435 API* que nous avons créé, et le compilateur *pepsy* de l'ISODE. Pour les deux APIs, nous présenterons leurs classes d'objets, ainsi que les fonctions qui y sont associées. Ensuite, nous présenterons le prototype, en discutant de ses fonctionnalités et de son architecture. Une partie du chapitre est consacré à la conformité du prototype, et aux problèmes que nous avons rencontrés lors de son développement.

Chapitre 1

Introduction Technique à l'EDI

1.1 Définition de l'EDI

EDI signifie *Electronic Data Interchange* ou en français *Echange de Données Informatisé*. Il peut être défini de la manière suivante:

> ''EDI est l'échange par des moyens de télécommunication de données structurées entre des applications informatiques distantes et sans intervention humaine.''

Le mot ''données'' signifie ici des documents commerciaux, bancaires, du service des douanes, du transport, des assurances, etc. Pour cette raison, des gens ont préféré le terme EBDI (*Electronic Business Data Interchange*) au terme EDI, estimant ce dernier peu précis. [GENILLOUD]

Pour comprendre la raison de l'introduction de l'EDI, il faut se rappeler la manière dont les gens s'échangent des documents jusqu'à présent. Depuis des siècles, cet échange se fait par le service de poste. Cela exige une grande consommation de papiers et, surtout, un délai d'attente qui peut durer des jours voire plus d'une semaine si l'échange se fait avec un pays étranger (voir figure 1-1). Alors que nous sommes à la fin de la vingtième siècle, ce délai d'attente paraît bien déraisonnable. Puisque nous disposons aujourd'hui des moyens de télécommunication performants et relativement fiables, et puisque la plupart des documents commerciaux sont aujourd'hui générés par des applications informatiques, alors pourquoi ne pas faire communiquer des documents directement entre des applications informatiques par des moyens de télécommunication (voir figure 1-2)? En procédant de la sorte, on peut réduire le délai d'attente de jours en secondes, ce qui est un gain très appréciable. C'est l'idée de départ qui a amené l'introduction de l'EDI. Contrairement à ce que beaucoup de gens pensent, cette révolution vient du monde économique et non du

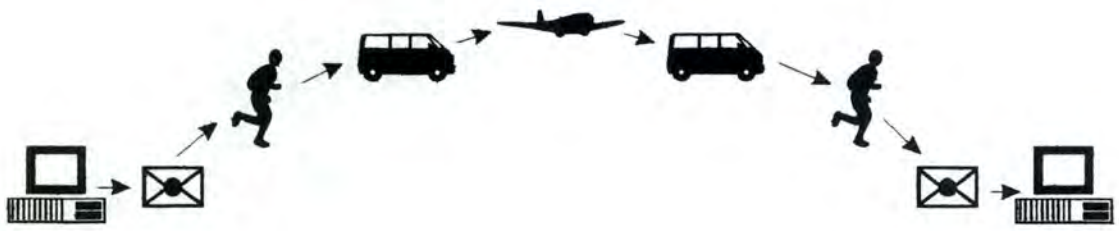

Figure 1-1 : Moyen classique de transfert de document

Figure 1-2 : Transfert de document par l'EDI

monde informatique.

En plus du gain de temps, l'EDI procure bien d'autres avantages tels que: [KENNY] [BLOCH]

-Livraison ''just-in-time'', menant à la réduction du niveau du stock.

-Réduction du coût du service de poste.

-Réduction du coût des papiers et de leur traitement. On estime le total mondial de ce coût à 1900 milliards de dollars en 1988. D'ailleurs, on qualifie souvent l'EDI de ''*paper-less trade*'' (commerce sans papier).

-Réduction des erreurs humaines. On estime en effet que les ressaisies d'un document commercial sont de 5 fois en moyenne et beaucoup d'erreurs de transcription risquent ainsi d'être commises.

-Amélioration de la sécurité, avec la confirmation de la réception d'un document. Bien sûr, on pourra toujours dire qu'aucun système de télécommunication n'est fiable à 100%, mais notre vie de tous les jours - et spécialement le service de poste - l'est-elle?

-Amélioration de la communication intra-compagnie.

Grâce à l'EDI, on estime une réduction de 25% sur les coûts administratifs liés à la production d'un produit, et des gains allant de 5% à 15% sur le prix des marchandises. [BLOCH]

L'avantage de l'EDI est donc évident. Mais pour que l'EDI puisse se généraliser, il faut que deux conditions soient remplies: d'une part, il faut un standard de communication universel pour que les utilisateurs puissent utiliser un même protocole d'échange; d'autre part, il faut un standard de représentation universel pour que les utilisateurs puissent reconnaître le format des messages envoyés par les autres. Ces deux conditions sont loin d'être satisfaites aujourd'hui. Nous allons donc voir comment on pourrait remédier à cette situation.

1.1.1 Standards de communication de l'EDI

Beaucoup de standards de communication sont utilisés aujourd'hui. Bien souvent, ce sont des protocoles privés utilisés par des partenaires commerciaux liés par un accord bilatéral.

A cause de cette situation, des sociétés se trouvent parfois dans l'impossibilité de communiquer avec d'autres sociétés à cause de l'incompatibilité des standards de communication utilisés. Il faut donc trouver un standard universel qui soit utilisé par tous les utilisateurs de l'EDI. Tout naturellement, on a pensé à X.400, une application du modèle OSI.

X.400 est le service de messagerie électronique de l'OSI. C'est un standard international qui est susceptible d'être implémenté sur tout système au monde. Pour cette raison, on l'a désigné comme la solution du futur pour le transfert des documents EDI. Le choix de X.400 pour l'EDI est en outre dû aux facteurs suivants: [GENILLOUD]

-X.400 sera utilisé par beaucoup de compagnies pour la messagerie électronique.

-X.400 est le premier protocole standard OSI à être disponible à la couche Application. Il peut transporter des messages de toute sorte de contenu et il sera supporté par tous les principaux vendeurs d'ordinateurs.

-X.400 définit également un service fourni par les administrations PTT. Un réseau public X.400 à l'échelle mondiale sera bientôt accessible dans tous les pays développés, l'adressage dans ce réseau sera très facile (à l'aide des services de répertoire X.500) et ce réseau sera connecté à d'autres services offerts par les PTT tels que télex, télétex, vidéotex ou fac-similé.

-X.400 est basé sur un concept " *store-and-forward* ". De ce fait, il n'est pas nécessaire d'établir une connexion avec le destinataire du message EDI. Le message EDI sera délivré même si le système du destinataire est momentanément coupé. (C'est la raison principale du choix de X.400 au détriment des autres applications OSI telles que FTAM)

-X.400 fournit des services de sécurité pour protéger le message EDI.

Les lecteurs peuvent aussi trouver dans [VANGUARD] des explications très précises sur le choix de X.400 comme standard de communication pour l'EDI.

X.400 deviendra très probablement le standard universel pour l'EDI, mais encore faut-il définir la manière dont X.400 doit être utilisé pour l'EDI. C'est chose faite par l'introduction en 1990 de la Recommandation X.435 par CCITT.

Nous discuterons de X.400 dans le chapitre 2 et de X.435 dans le chapitre 3.

1.1.2 Standards de représentation de l'EDI

De nombreux standards de représentation des messages EDI sont utilisés aujourd'hui. Ce sont soit des standards privés utilisés dans des domaines bien précis, soit des standards nationaux utilisés à une plus grande échelle. Les plus connus parmi eux sont:

-ODETTE (standard utilisé dans l'industrie de l'automobile)
-SWIFT (standard utilisé pour le transfert bancaire)
-UN-TDI (standard utilisé essentiellement en Europe)
-TRADACOMS (standard britannique)
-ANSI X.12 (standard américain)

L'existence des nombreux standards est dû à la différence de cultures, de langages, de pratiques administratives et commerciales des gens désireux d'utiliser l'EDI. Mais à cause de cette situation, deux sociétés utilisant des standards de représentation différents se trouvent dans l'impossibilité de s'échanger des messages EDI entre eux. Il faut donc impérativement définir un standard universel qui soit accepté par tout le monde si l'on veut voir l'EDI se généraliser avec succès. C'est chose faite aujourd'hui avec la définition par les Nations Unies du standard **EDIFACT.**

Pour comprendre le processus qui a mené un monde EDI désordonné au standard EDIFACT, il est certainement utile de présenter l'historique de l'EDI depuis une trentaine d'années.

Historique de l'EDI [BLOCH]

C'est aux Etats-Unis, dans les années soixante, qu'ont débuté les premières expériences d'échanges de données commerciales et du transport par voies électroniques. Les promoteurs en furent les compagnies de transport ferroviaires et de grandes compagnies telles que DuPont De Nemours, General Motors, etc.

En 1961 fut créé un premier groupe de travail (*Working Party*) pour la normalisation de l'EDI au sein de l'UN/ECE Committee on Development of Trade. (UN/ECE signifie *United Nations Economic Commission for Europe*)

En 1979, l'ANSI (*American National Standards Institute*) basé à New York, charge l'ASC (*Accredited Standards Committee*) X.12 du développement pour les Etats-Unis d'une norme d'usage universel. Ce sera le standard ANSI X.12.

En 1979, l'UN/ECE qui siège à Genève confédère une vingtaine de pays et a comme objectif l'élaboration de structures de données, de contenus et d'une syntaxe normalisée appropriés à l'EDI.

Entre 1979 et 1985, ANSI X.12 développe un standard orienté commerce intérieur, tandis que l'UN/ECE s'intéresse au commerce international avec une approche sensiblement différente de l'ANSI X.12 et publie GTDI (*Guidelines for Trade Data Interchange*), appelé également UN-TDI.

En 1984, ANSI X.12 et TDCC (*Transportation Data Coordinating Committee*) créent

un comité d'harmonisation, le JEDI (*Joint EDI*). Placé sous l'égide des Nations Unies, l'UN-JEDI est désormais le siège des réunions d'experts américains et européens pour la convergence des normes d'échanges EDI.

En 1985 et 1986, les réunions d'experts pour l'harmonisation se poursuivent pour la convergence des standards ANSI X.12, UN-TDI et autres standards privés. Le groupe de travail WP4 encourage les experts nord-américains et européens à mettre au point une norme unique pour l'échange des données commerciales, soutient les propositions de l'UN-JEDI et préconise la confection de messages standard.

En 1987, le groupe de travail entérine l'acronyme **EDIFACT** et nomme trois rapporteurs, un pour l'Europe de l'Ouest (EDIFACT Board), un pour l'Amérique du Nord (ASC X.12) et un pour l'Europe de l'Est (EDIFACT Committee).

Voilà la longue histoire qui a mené à la définition du standard EDIFACT, qui est destiné à être le standard de représentation mondial pour l'échange EDI. Aujourd'hui, l'EDIFACT commence à se généraliser dans les pays européens, mais il a encore du mal à s'imposer dans le reste du monde. C'est notamment la cas des Etats-Unis, où l'utilisation du standard ANSI X.12 est encore très répandue. Cependant, le gouvernement américain a officiellement appuyé l'utilisation du standard EDIFACT, et le service des douanes américain a récemment décidé d'utiliser exclusivement l'EDIFACT pour tous ses échanges EDI. Cela devrait aider EDIFACT à se généraliser dans ce pays et on estime dans [GENILLOUD] qu'un délai de cinq à dix ans sera nécessaire pour la convergence de l'ANSI X.12 vers l'EDIFACT.

1.1.3 *Value Added Network*

Aujourd'hui, les compagnies utilisant des standards de communication et de représentation différents parviennent néanmoins à communiquer grâce aux services de VAN (*Value Added Network*).

Le VAN est une compagnie qui fournit des VADS (*Value Added Data Services*) à des sociétés, entre autres celles utilisatrices de l'EDI. Le VAN possède un ou plusieurs grands centres d'ordinateurs, supporte plusieurs protocoles de communication et fournit des boîtes aux lettres logiques aux clients.

Les VADS fournis aux clients sont: [GENILLOUD]

-Conversion des protocoles de communication.

-Conversion des syntaxes et des formats.

-Boîtes aux lettres pour stocker des messages en entrée.

-Rapports sur le statut des messages.

-Autres services (sécurité, statistiques, ...)

-**Services spécifiques à l'EDI**: contrôle de la syntaxe, conversion des formats de message EDI, recherches sélectives des messages EDI, ... etc.

Grâce au VAN, deux sociétés utilisant des protocoles de communication et des représentations différents peuvent entrer en communication. Prenons le cas où une société A et une société B utilisant des protocoles de communication différents mais connectées toutes deux à un VAN. Quand A veut envoyer un message EDI à B, il l'envoie d'abord au VAN. Ce dernier fait la conversion des protocoles de communication et des formats du message, puis place le résultat dans la boîte aux lettres de B, dans laquelle B pourra venir retirer le message. Le problème de l'incompatibilité des standards est ainsi contourné comme le montre la figure 1-3. (Il faut cependant remarquer que beaucoup considèrent que la conversion des formats n'est pas possible à cause des problèmes d'interprétation.)

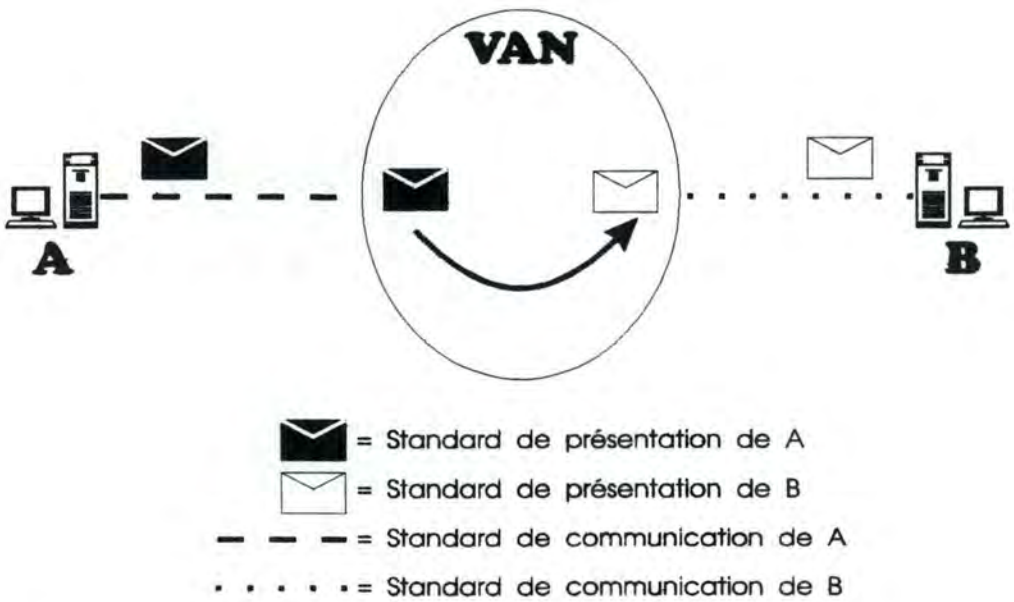

Figure 1-3 : Traduction des standards de communication et de représentation par le VAN

Les VANs les plus connus aux Etats-Unis sont certainement IBM, GEISCO, EDS et McDonell Douglas. Les VADS fournis par ces VANs sont très coûteux. De plus, sans doute pour protéger leur profit, les différents VANs ne sont pas interconnectés entre eux. Cela oblige souvent des sociétés à se souscrire aux services de plusieurs VANs pour pouvoir atteindre le plus grand nombre de clients potentiels possibles.

Malgré le coût élevé, les VANs connaissent aujourd'hui un franc succès dans le monde de l'EDI. La future généralisation de X.400 étant prévisible, la plupart des VANs s'intégreront sans doute dans un réseau X.400 tout en fournissant des services EDI supplémentaires. Dans le but de promouvoir X.400, le gouvernement britannique

prévoit d'ailleurs l'obligation de supporter X.400 pour les VANs fournissant des services de messagerie. [GIFKINS]

1.2 Description du standard EDIFACT

EDIFACT signifie *Electronic Data Interchange For Administration, Commerce and Transport.* Il a été défini à partir des standards UN-TDI et ANSI X.12 tout en gardant les meilleures caractéristiques de ceux-ci. On l'appelle parfois UN/EDIFACT, le préfixe ''UN'' indiquant simplement qu'il s'agit d'une Recommandation de l'UN/ECE.

Le standard EDIFACT est composé d'une grammaire et d'un vocabulaire. Il est défini dans les documents suivants:

Grammaire

ISO 9735 - EDIFACT Syntax Rules :
Définition des règles de syntaxe de l'EDIFACT au niveau Application.

Vocabulaire

UN/EDIFACT Code Lists (EDCL) :
Définition des codes associés aux éléments de donnée simples.

UN/EDIFACT Data Elements Directory (EDED) :
Définition des éléments de donnée simples. C'est un extrait du document **ISO 7372 - Trade Data Elements Directory (TDED) .**

UN/EDIFACT Composite Data Elements Directory (EDCD) :
Définition des éléments de donnée composites.

UN/EDIFACT Standard Data Segments Directory (EDSD) :
Définition des segments de donnée standard utilisés pour les messages EDIFACT.

UN/EDIFACT Data Messages Directory (EDMD) :
Définition des messages standard EDIFACT spécifiés par l'UN/ECE (UNSMs).

UN/EDIFACT Syntax Implementation Guide :
Explications plus détaillées sur l'implémentation des règles de syntaxe (*syntax rules*) de l'ISO 9735.

UN/EDIFACT Message Design Guidelines :
Conseils à ceux qui voudraient créer de nouveaux messages EDIFACT.

Tous ces documents sont regroupés dans un document-répertoire nommé ''*United Nations Trade Data Interchange Directory*'' ou UNTDID (voir [UNTDID]). Il sort

deux versions d'UNTDID par an: la version ''.1'' et la version ''.2'' (par exemple 90.1 et 90.2 pour l'année 1990). La version ''.1'' paraît en novembre, c'est une version d'essai (''*trial version*'') qui a une période de stabilité d'un an. Après cette période, elle accèdera au statut ''.2'' avec éventuellement quelques changements. Grâce à la version ''.1'', on peut à l'avance avoir un aperçu de ce que sera la future version ''.2''. La version ''.2'' paraît en mai, c'est une version dite définitive qui a une période de stabilité de trois ans.

Deux niveaux de syntaxe sont définis dans ISO 9735: les niveaux A et B. La différence entre les deux niveaux est le jeu de caractères (notamment les caractères de séparation) employé. Comme le niveau A est de loin le plus utilisé, tous nos exemples dans ce chapitre seront codés selon la syntaxe du niveau A.

1.2.1 Structure générale de l'interchange EDIFACT

Le fichier EDIFACT échangé entre deux partenaires est appelé ''l' **interchange**

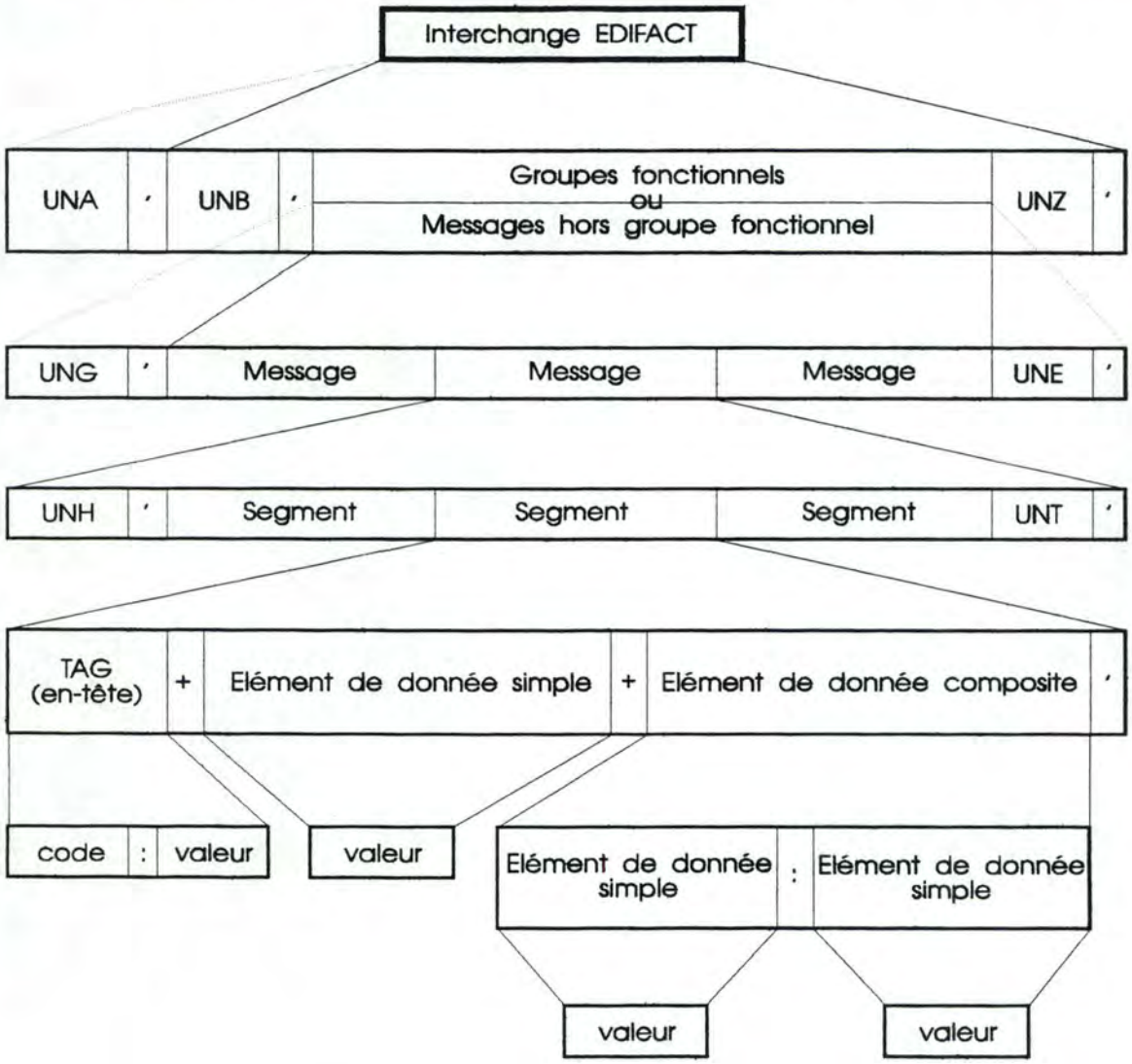

Figure 1-4 : Structure générale de l'interchange EDIFACT

EDIFACT''. L'interchange est un ensemble structuré et hiérarchique de messages EDI et de segments de service (''UNx'') comme le montre la figure 1-4.

Voici la signification de cette figure:

Un interchange EDIFACT est composé de **groupes fonctionnels** ou de **messages** indépendants sans groupe fonctionnel.

Un groupe fonctionnel comporte un ensemble de **messages**. Il sert à regrouper des messages du même type (ex.: des factures) ou des messages de la même sous-adresse (l'adresse principale se trouvant dans l'en-tête de l'interchange). L'utilisation du groupe fonctionnel n'est pas obligatoire, bien qu'elle soit quasi systématique en Amérique du Nord.

Un message est composé de **segments** selon une structure hiérarchique. Les messages standard définis par l'UN/ECE (UNSMs) se trouvent dans le document EDMD.

Un segment est composé d' **éléments de donnée simples** et/ou d' **éléments de donnée composites**. Il y a deux types de segment: les segments de service (segments ''UNx'') et les segments de données. La définition des segments se trouve dans le document EDSD.

Un élément de donnée composite est composé d' **éléments de donnée simples**. La définition des éléments de donnée composites se trouve dans le document EDCD.

Un élément de donnée simple est un champ de valeur. Il peut être de type alphabétique, numérique, alphanumérique ou '' **code** identifiant''. La définition des éléments de donnée simples se trouve dans le document EDED.

Un code est l'identifiant d'une valeur choisie dans une liste de valeurs prédéfinies pour un élément de donnée simple particulier. Les listes des codes se trouvent dans le document EDCL.

Nous allons décrire ces composants de l'interchange EDIFACT en adoptant une approche ''bottom-up'', c'est-à-dire en partant du plus simple (code) pour arriver au plus complexe (message). Pour chacun de ces composants, nous donnerons un exemple concret, avec le codage associé défini par ISO 9735. Nous laissons toutefois la description du **message** au paragraphe suivant (1.3) pour y discuter de l'important concept du diagramme de branchement.

1.2.2 Code

Pour les éléments de donnée simples de type ''code identifiant'', les valeurs possibles sont déterminées à l'avance par l'ISO. Ces valeurs sont alors codées et regroupées dans une liste des codes. Pour ces éléments de donnée, on est obligé de choisir un code dans la liste et on n'est pas autorisé à utiliser une valeur non-prédéfinie.

Prenons un exemple bien concret. L'élément de donnée simple '' *Measure unit*

specifier'' est de type ''code identifiant'', il sert à spécifier l'unité de mesure utilisée. On trouve donc dans le document EDCL la (longue) liste des codes pour cet élément de donnée, liste dans laquelle l'ISO a répertorié toutes les unités de mesure imaginables et les a codées. En voici un petit extrait:

```
6411  Measure unit specifier

Unit name  Code

Gram          GRM
Kilogram      KGM
Milligram     MGM
Day           DAY
Week          WEE
Year          ANN
Litre         LTR
Gallon        GLI
Cubic meter   MTQ
.
.
.
```

Le nombre 6411 est la référence de l'élément de donnée, et ''Measure unit specifier'' en est le nom. Suit alors une liste de valeurs possibles, avec leur code correspondant. Par exemple, si l'unité de mesure utilisée est le kilogramme, alors on doit prendre comme valeur le code KGM.

1.2.3 Elément de donnée simple

Un élément de donnée simple est un élément de donnée qui ne comporte qu'un seul champ de valeur. C'est par exemple le cas de l'élément de donnée '' *Measure unit specifier*'' déjà discuté plus haut. Voyons comment il est défini dans le document EDED:

```
6411  Measure unit specifier

Desc: Indication of the unit of measurement in which weight (mass), capacity,
length, area, volume or other quantity is expressed.

Repr: a3  Min: 3    Max: 3    Datatype: id
```

Tout comme dans la liste des codes, le nombre 6411 est la référence de l'élément de donnée, et ''Measure unit specifier'' en est le nom. ''Desc'' est la description en anglais de l'élément de donnée. ''Repr'' indique la représentation, qui est ici alphanumérique de longueur fixe 3. ''Min'' et ''Max'' indiquent la longueur minimale et maximale, qui sont bien entendu 3 dans ce cas-ci. Et ''Datatype'' indique le type, qui est ici ''id'' qui signifie ''code identifiant''.

La raison de la double présence de l'élément de donnée '' *Measure unit specifier*'' dans EDED et dans EDCL est la suivante: EDED explique la signification de l'élément de donnée, avec la spécification de la représentation et du type. Si

l'élément de donnée est du type ''code identifiant'', alors on peut retrouver dans EDCL la liste des codes qui le concerne. Il n'y a donc pas de redondance.

Prenons un autre exemple extrait d'EDED:

```
6060  Quantity

Desc: Numeric value of quantity

Repr: n..15     Min: 1     Max:15     Datatype: n
```

C'est l'élément de donnée simple '' *Quantity*'' qui porte la référence 6060. Il a une représentation numérique de longueur minimale 1 et de longueur maximale 15. Son type est numérique et non ''code identifiant'', cela signifie que '' *Quantity*'' ne se retrouve pas dans EDCL et que sa valeur peut être déterminée librement.

Le codage d'un élément de donnée simple est facile: on met sa valeur telle quelle, sans guillemet ou autres préfixes/suffixes pour les éléments de donnée du type alphabétique ou alphanumérique. Par exemple, si la valeur de '' *Quantity*'' est 100, alors son codage est tout simplement

100

Si la valeur de '' *Measure Unit Specifier*'' est le kilogramme, alors son codage est

KGM

1.2.4 Elément de donnée composite

Un élément de donnée composite est un regroupement ordonné d'éléments de donnée simples formant un ensemble cohérent [BLOCH]. Autrement dit, c'est un élément de donnée qui a plus d'un champ de valeur.

Prenons l'exemple suivant extrait d'EDCD:

```
C186  QUANTITY INFORMATION

Desc: Quantity information in a transaction, qualified when relevant.

Cont:
6063  Quantity qualifier              C    an..3   id   1    3
6060  Quantity                        M    n..15   n    1    15
6411  Measure unit specifier          C    an..3   id   1    3
```

Il s'agit de l'élément de donnée composite '' *QUANTITY INFORMATION*'' qui porte la référence C186. Il est composé de trois éléments de donnée simples présentés en colonnes: la première colonne indique la référence, la deuxième indique le nom, la troisième indique si le composant est obligatoire (M pour '' *mandatory*'') ou facultatif (C pour '' *conditional*''), la quatrième indique la représentation, la cinquième indique le type, et les deux dernières indiquent les longueurs minimale et maximale.

Le premier composant - l'élément de donnée simple " *Quantity qualifier*" - est de type "code identifiant", il donne la signification spécifique à une quantité. Par exemple, le code 21 signifie "quantité commandée", et le code 47 signifie "quantité facturée". Quant aux deux autres composants, nous les avons déjà décrits plus haut dans ce paragraphe. Les trois forment l'élément de données composite " *QUANTITY INFORMATION*" qui fournit l'information de quantité dans une transaction.

Pour un élément de donnée composite, ISO 9735 spécifie la règle de codage suivante: on présente les valeurs des composants dans leur ordre d'apparition dans la définition, séparées par le caractère ':' (deux-points). Par exemple, pour une quantité facturée de 100 kilogrammes, on a " *Quantity qualifier*"=47, "*Quantity*"=100 et "*Measure unit spécifier*"=KGM. Le codage de " *QUANTITY INFORMATION*" est donc:

47:100:KGM

Il faut remarquer que les composants " *Quantity qualifier*" et "*Measure unit spécifier*" sont facultatifs. Si l'un ou l'autre n'est pas présent, il est omis dans le codage, mais le séparateur ":" doit y être maintenu pour qu'on puisse détecter son absence. Cette règle est nommée "Exclusion des composants des éléments de donnée composites par omission".

Nous reprenons le codage ci-dessus. Selon les cas où le premier, le dernier ou les deux composants facultatifs sont absents, on a les codages suivants:

:100:KGM
47:100:
:100:

On trouvera dans 1.2.5 (la règle 4) d'autres précisions concernant le codage d'un élément de donnée composite.

1.2.5 Segment

Un segment est une suite ordonnée d'éléments de donnée simples ou composites. Le regroupement des éléments de donnée au sein des segments facilite la construction des messages EDIFACT, rend plus aisée la conception et la consultation des bases de données et facilite la saisie et l'édition des informations des documents [BLOCH]. Les éléments de donnée regroupés dans un segment sont destinés à être traités en même temps et sont relatifs aux mêmes fonctions.

Prenons l'exemple suivant extrait de EDSD:

```
QVA  QUANTITY VARIANCES

Function: To specify item details relating to variations between ordered/shipped
and invoiced quantities.
```

C186	QUANTITY INFORMATION	M				
6063	Quantity qualifier	C	an..3	id	1	3
6060	Quantity	M	n..15	n	1	15
6411	Measure unit specifier	C	an..3	id	1	3
4221	SHIPMENT/ORDER DISCREPANCY, CODED	C	an..2	id	1	2
6064	QUANTITY DIFFERENCE	C	n..15	n	1	15
C262	REASON FOR CHANGE	C				
4295	Change reason, coded	C	an..2	id	1	2
4294	Change reason	C	an..35	an	1	35

C'est le segment "*QUANTITY VARIANCES*" qui porte comme TAG (étiquette de référence) QVA. La rubrique "function" explique clairement qu'il s'agit d'un segment qui sert à spécifier les détails relatifs à des variations entre la quantité commandée/envoyée et la quantité effectivement facturée (dues à une rupture de stock, par exemple). Le segment comporte deux éléments de donnée simples et deux éléments de donnée composites. Dans le cas de ces derniers, les composants sont rappelés dans la définition du segment. La présentation est faite de la même manière que pour les éléments de donnée composites vus plus haut, c'est-à-dire en colonnes. Bien entendu, les colonnes gardent les mêmes significations.

L'élément de donnée composite "*QUANTITY INFORMATION*" a déjà été discuté. L'élément de donnée simple "*SHIPMENT/ORDER DISCREPANCY, CODED*" est un code qui indique les dispositions à prendre suite aux variations de quantité (par exemple, BP = "Envoi partiel, le restant suivra"). L'élément de donnée simple "*QUANTITY DIFFERENCE*" indique la différence de quantité, et l'élément de donnée composite "*REASON FOR CHANGE*" comporte deux éléments de donnée simples: "*Change reason, coded*" est un code qui indique la raison du changement (par exemple, QO = "Changement dû à la quantité commandée"), et "*Change reason*" contient des commentaires supplémentaires concernant le changement.

Pour un segment, ISO 9735 spécifie la règle de codage de base suivante: le TAG (étiquette) du segment doit apparaître en premier lieu. On présente ensuite les valeurs des différents éléments de donnée dans leur ordre d'apparition dans la définition, séparé par le caractère "+" (plus). Ce caractère sert également à séparer le TAG du premier élément de donnée, sauf pour le segment de service UNA qui sert précisément à définir les caractères de séparation (voir 1.2.6). Finalement, le segment doit se terminer par le caractère ' (apostrophe).

Nous allons prendre un cas concret. Imaginons qu'une société A commande à une société B 120 kilos d'un produit quelconque. La commande étant très importante, B ne peut livrer que 100 kilos en ce moment. B envoie alors une facture EDIFACT de 100 kilos à A, avec le segment QVA mis aux valeurs suivantes:

"*QUANTITY INFORMATION*":
"*Quantity qualifier*"=47 (= Quantité facturée)
"*Quantity*"=100
"*Measure unit specifier*"=KGM (= Kilogramme)
"*SHIPMENT/ORDER DISCREPANCY, CODED*"=BP (= Envoi partiel)
"*QUANTITY DIFFERENCE*"=20 (Différence = 20 kilos)
"*REASON FOR CHANGE*":

''*Change reason, coded*''=QO (= Dû à la quantité commandée)
''*Change reason*''=Stock insuffisant (Commentaire)

Le codage du segment QVA est donc:

QVA+47:100:KGM+BP+20+QO:Stock insuffisant'

ISO-9735 spécifie d'autres règles de codage de segment pour les cas où des composants facultatifs ou des éléments de donnée facultatifs sont absents. Nous allons décrire ces règles, en prenant comme exemple le codage ci-dessus:

1) Exclusion des éléments de donnée par omission

Si un élément de donnée facultatif est absent et s'il est suivi d'un autre élément de donnée, alors il n'apparaît pas dans le codage mais sa position doit être indiquée par la rétention de son caractère de séparation ''+''.

Par exemple, si on supprime l'élément de donnée '' *SHIPMENT/ORDER DISCREPANCY, CODED*'', alors on a le codage suivant:

QVA+47:100:KGM++20+QO:Stock insuffisant'

2) Exclusion des éléments de donnée par troncation

Si un ou plusieurs éléments de donnée facultatifs à la fin d'un segment sont absents, alors le segment peut être tronqué immédiatement par le caractère de fin de segment ''''.

Par exemple, si on supprime l'élément de donnée composite '' *REASON FOR CHANGE*'', alors on a le codage suivant:

QVA+47:100:KGM+BP+20'

On remarque que dans ce cas-ci, la rétention du caractère de séparation ''+'' n'est plus nécessaire.

3) Exclusion des composants des éléments de donnée composite par omission

(Déjà expliquée à la fin de 1.2.4)

4) Exclusion des composants des éléments de donnée composite par troncation

Si un ou plusieurs composants facultatifs à la fin d'un élément de donnée composite sont absents, alors l'élément de donnée composite peut être tronqué immédiatement par le caractère de séparation ''+''.

Par exemple, si on supprime le composant " *Measure unit specifier*" de l'élément de donnée composite " *QUANTITY INFORMATION*", alors on a le codage suivant:

QVA+47:100+BP+20+QO:Stock insuffisant'

On remarque que dans ce cas-ci, la rétention du caractère de séparation ":" n'est plus nécessaire.

Un dernier exemple: si on supprime tous les composants et tous les éléments de donnée facultatifs, alors on le codage suivant:

QVA+:100'

La rétention des caractères de séparation ":" et "+" n'est plus nécessaire.

1.2.6 Segment de service

Il existe deux types de segment: les segments de données et les segments de service. Les segments de données sont ceux qui contiennent les informations internes au message (ex.: noms et adresses, dates, articles, quantités, montants, valeurs, ... etc); le segment QVA en est un exemple. Les segments de service définissent le cadre et les caractéristiques de l'interchange EDIFACT. De plus, ils servent de délimiteurs pour séparer les interchanges, les groupes fonctionnels et les messages.

Les segments de service sont au nombre de huit. Ils portent tous un TAG qui commence par les lettres "UN" et ils sont définis dans EDSD de la même manière que pour les segments de données. Voici une brève description de ces segments de service:

Segment UNA: *Service string advice* (facultatif)

Le segment UNA permet de préciser les caractères de service utilisés. Ce sont les caractères de séparation, d'indication et d'échappement. La figure 1-5 montre les caractères de service par défaut pour le niveau A de la syntaxe EDIFACT.

Fin de segment	' (apostrophe)
Séparateur entre éléments de donnée du segment, et entre le TAG et le premier élément de donnée du segment	+ [plus]
Séparateur entre éléments constitutifs d'un élément de donnée composite	: [deux-points]
Caractère d'échappement	? [point d'interrogation]

Figure 1-5 : Caractères de service par défaut du codage EDIFACT

Le caractère d'échappement, lorsqu'il précède un autre caractère, précise que ce dernier n'est pas un caractère de service mais bien un caractère de donnée. Il permet de ne pas avoir de confusion lorsqu'un caractère de service est utilisé dans les données par coïncidence. Pour reprendre l'exemple du segment QVA déjà discuté, si on avait décidé de donner la valeur suivant à l'élément de donnée *''Change reason''*:

''Change reason'' = Raison : stock insuffisant

Alors le codage du segment serait devenu:

QVA+47:100:KGM+BP+20+QO:Raison ?: stock insuffisant'

Et au cas où le caractère d'échappement est lui-même présent dans les données, on n'a qu'à le doubler dans le codage (''??'').

Le segment UNA, s'il est présent au début de l'interchange, permet de modifier ces caractères de service utilisés par défaut. UNTDID conseille cependant de ne les modifier qu'en cas d'extrême nécessité, comme par exemple lorsque les caractères de service par défaut apparaissent trop souvent dans les données mêmes du message. C'est notamment le cas d'un message EDIFACT qui contient des formules mathématiques, où le caractère ''+'' est très utilisé. La présence du segment UNA permet alors d'imposer d'autres caractères de service et nous évite ainsi de devoir employer trop souvent le caractère d'échappement.

Segment UNB : *Interchange header* (obligatoire)

Le segment UNB est le segment d'en-tête de l'interchange. Il sert à initialiser, identifier et préciser le cadre de l'interchange. Les principaux éléments de donnée du segment UNB sont:

-L'identification et le numéro de version de la syntaxe.

-L'identification de l'émetteur de l'interchange. (L *'EDI Name*. Voir 4.2.1)

-L'identification du récepteur de l'interchange. (L' *EDI Name*)

-La date et l'heure de préparation de l'interchange.

-La référence de contrôle de l'interchange.

Segment UNZ : *Interchange trailer* (obligatoire)

Le segment UNZ est le segment de fin de l'interchange. Il sert à terminer l'interchange et il peut être utilisé pour contrôler l'achèvement de l'interchange. Les éléments de donnée du segment UNZ sont:

-Le compteur du nombre de messages (ou de groupes fonctionnels) contenus

dans l'interchange. Il permet de vérifier si tous les messages (ou groupes fonctionnels) sont bien reçus.

-La référence de contrôle de l'interchange. Cette référence est identique à celle du segment UNB.

Segment UNG : *Functional group header* (facultatif)

Le segment UNG est le segment d'en-tête du groupe fonctionnel, il sert à identifier le groupe fonctionnel. Bien que ce segment soit facultatif, il est toujours présent dans les échanges en Amérique du Nord. C'est un facteur important à tenir compte pour les interchanges échangés avec les pays de ce continent. Les principaux éléments de donnée du segment UNG sont:

-L'identification du groupe fonctionnel.

-L'identification de l'émetteur.

-L'identification du récepteur.

-La référence de contrôle du groupe fonctionnel.

Segment UNE : *Functional group trailer* (facultatif)

Le segment UNE est le segment de fin du groupe fonctionnel. Il sert à terminer le groupe fonctionnel et il peut être utilisé pour contrôler l'achèvement du groupe fonctionnel. Les éléments de donnée du segment UNE sont:

-Le compteur du nombre de messages contenus dans le groupe fonctionnel. Il permet de vérifier si tous les messages sont bien reçus.

-La référence de contrôle du groupe fonctionnel. Cette référence est identique à celle du segment UNG.

Segment UNH : *Message header* (obligatoire)

Le segment UNH est le segment d'en-tête du message, il sert à identifier le message. Les principaux éléments de donnée du segment UNH sont:

-Le numéro de référence du message.

-L'identification du message. Précisions sur le type (ex.: bon de commande), la version (ex.: 90.2), ... etc.

Segment UNT : *Message trailer* (obligatoire)

Le segment UNT est le segment de fin du message. Il sert à terminer le message et il peut être utilisé pour contrôler l'achèvement du message. Les éléments de donnée du segment UNT sont:

-Le compteur du nombre de segments contenus dans le message, y compris les segments UNH et UNT. Il permet de vérifier si tous les segments sont bien reçus.

-Le numéro de référence du message. Ce numéro est identique à celui du segment UNH.

Segment UNS : *Section contrôle* (facultatif, dépendant du type de message)

Le segment UNS est le segment de séparation entre les sections du message. En effet, certains types de message ont leurs segments de donnée séparés en trois sections: l'en-tête (*header*), le détail (*detail*) et le résumé (*summary*). C'est notamment le cas du message INVOIC (facture). Des segments UNS servent alors de séparateurs entre ces différentes sections.

Nous avons mentionné le caractère facultatif du segment UNS, car sa présence dans un message dépend de la définition de celui-ci. Certains messages ne l'utilisent pas, n'ayant pas divisé leurs segments en sections. Mais lorsqu'il est utilisé par le message, alors il est toujours **obligatoire.**

Le segment UNS comporte un seul élément de donnée, qui est l'identification de la section.

TAG	Intitulé du segment	Statut
UNA	Avis de caractères de service	Facultatif
UNB	En-tête d'interchange	Obligatoire
UNG	En-tête de groupe fonctionnel	Facultatif
UNH	En-tête de message	Obligatoire
	(Segments de donnée de la section d'en-tête)	
UNS	Segment de séparation de section	Facultatif
	(Segments de donnée de la section de détail)	
UNS	Segment de séparation de section	Facultatif
	(Segments de donnée de la section de résumé)	
UNT	Fin de message	Obligatoire
UNE	Fin de groupe fonctionnel	Facultatif
UNZ	Fin d'interchange	Obligatoire

Figure 1-6 : Disposition des segments de service dans l'interchange EDIFACT

Nous avons maintenant terminé la discussion sur les segments de service dans un interchange EDIFACT. La disposition des ces segments est résumée dans la figure 1-6.

1.3 Message EDIFACT

Un message est une suite ordonnée de segments selon une structure hiérarchique à plusieurs niveaux d'imbrication. La définition des messages standard, c'est-à-dire la description des segments constituants des messages standard, se trouve dans le document EDMD.

Beaucoup de messages sont en étude à l'UN/ECE. Selon leur état d'avancement, ils portent un statut allant de 0 à 2. Voici la signification de ces statuts:

0 = Document provisoire (*Draft document*).
P = Proposition provisoire (*Draft proposal*) agréée par les rapporteurs EDIFACT.
1 = Document provisoire pour essai formel (*Draft for formal trial*).
2 = Recommandation (message standard).

Un message doit passer ces différents stades d'étude avant de devenir un message standard, c'est-à-dire de statut 2. Lorsqu'un message atteint le statut 2, il est publié dans le document EDMD. Les messages de statut 2 les plus connus sont certainement la facture et le bon de commande. Chaque type de message porte un identifiant de six lettres, par exemple INVOIC pour la facture et ORDERS pour le bon de commande. Cet identifiant doit se trouver dans le segment UNH lors de l'envoi d'un message pour indiquer au récepteur le type de message.

En septembre 1991, 19 messages de statut 2 sont entérinés à l'UN/ECE et publiés. 16 messages de statut 1 et 35 messages de statut 0 et P sont en étude à l'UN/ECE.

Un message peut être défini de deux manières: par un diagramme de branchement, ou par une table de segments.

1.3.1 Diagramme de branchement

Le diagramme de branchement est une représentation graphique qui définit un message EDIFACT en montrant les segments constituants et leur imbrication.

La figure 1-7 (repris de [BLOCH]) montre un diagramme de branchement qui définit un message fictif. Il faut lire le diagramme de gauche à droite. Chaque rectangle du dessin représente un segment, avec des précisions sur le TAG, le statut (C=facultatif, M=obligatoire) et le nombre de répétitions du segment. Par exemple, le segment AAA est un segment obligatoire non répétitif, et le segment HHH est un segment facultatif qui peut se répéter au maximum 5 fois. Il est à noter que la plupart des segments de cette figure sont fictifs. Comme tout message, notre message fictif commence par un segment UNH et se termine par un segment UNT; tous les autres segments étant des segments de données qui contiendraient les informations du message.

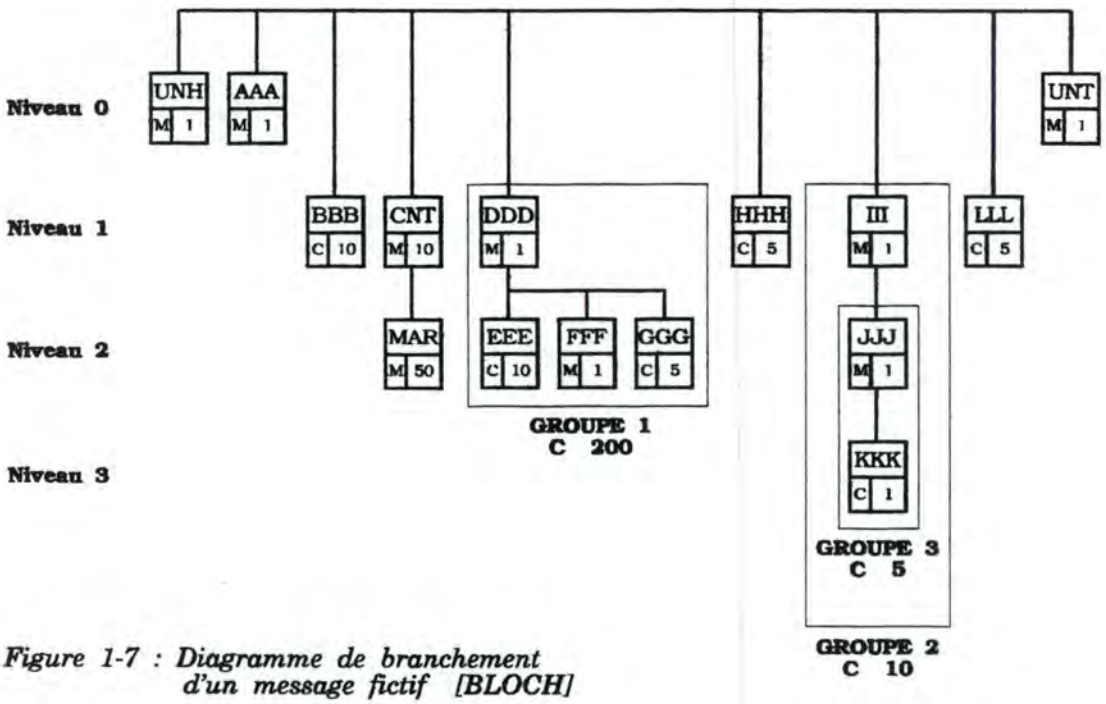

Figure 1-7 : Diagramme de branchement d'un message fictif [BLOCH]

Les segments sont présentés en plusieurs niveaux d'imbrication. Au niveau 0 se trouvent uniquement les segments non-répétitifs qui ne font pas partie d'une structure d'imbrication. Pour prendre un cas d'imbrication, regardons les segments CNT et MAR. Contrairement aux autres segments, ces deux segments ne sont pas fictifs et nous l'avons fait expressément pour bien montrer l'intérêt de l'imbrication. Un rapide coup d'oeil dans le document EDSD nous permet de connaître la signification de ces segments et d'imaginer la situation suivante: le segment CNT contiendrait les données décrivant le poids total et la valeur totale d'un conteneur d'une expédition de marchandises, et le segment MAR contiendrait les données relatives au poids et à la valeur d'un seul type de marchandise pouvant être stocké dans le conteneur. On peut avoir au maximum 10 conteneurs et 50 types de marchandise par conteneur. Le diagramme implique que dans le codage du message, après l'apparition d'un segment CNT, il y aura 1 à 50 segments MAR qui suivent, le nombre de segments MAR étant le nombre de types de marchandise qui sont stockés dans le conteneur en question. Le segment CNT peut apparaître de 1 à 10 fois, le nombre d'apparitions étant égal au nombre de conteneurs expédiés. Nous voyons ainsi l'intérêt de l'imbrication: le segment MAR est imbriqué dans le segment CNT, car le premier est dépendant du dernier.

Les segments EEE, FFF et GGG sont imbriqués dans le segment DDD, leur père commun, appelé également **segment de contrôle**. Les quatre segments font partie d'un groupe, le groupe n°1, qui est facultatif avec un nombre de répétitions de 200. Cela implique que dans le codage du message, après l'apparition du segment DDD, il y aura 0 à 10 segments EEE, suivis nécessairement d'1 segment FFF, suivi de 0 à 5 segments GGG. Ce groupe n°1 peut lui-même apparaître de 0 à 200 fois consécutivement dans le message codé.

Le principe semble identique à celui des segments CNT et MAR, mais à quoi sert la notion du groupe? Pour répondre à cette question, regardons d'abord la figure 1-8.

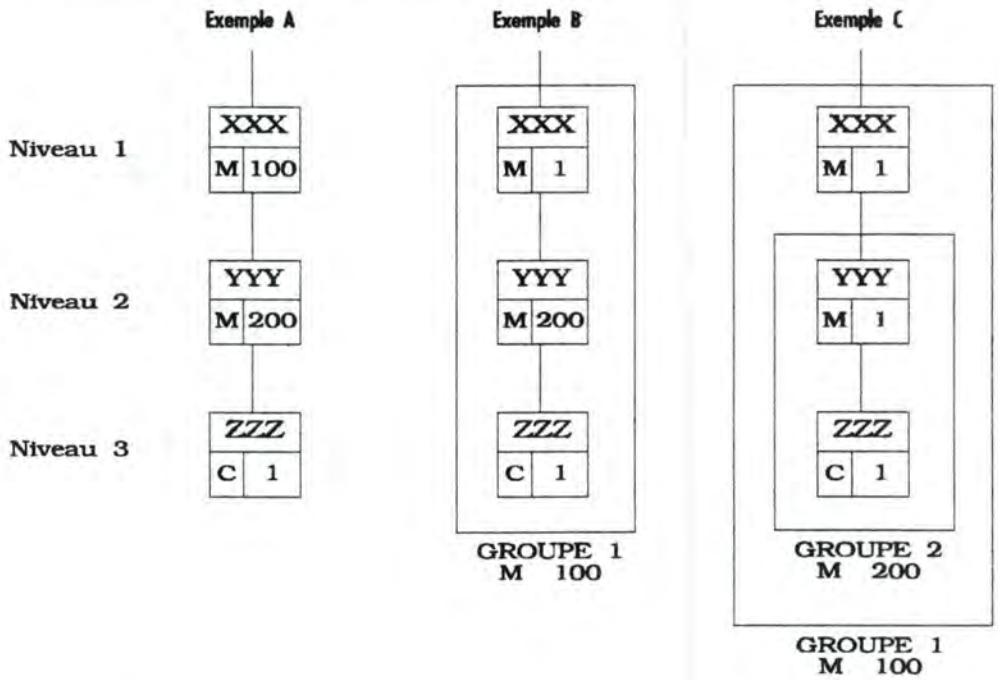

Figure 1-8 : Trois diagrammes différents décrivant un même concept

Dans cette figure sont présentés trois diagrammes différents dans leur représentation mais identiques au point de vue sémantique. On voit bien que les exemples B et C compliquent inutilement le diagramme sans rien y ajouter. En effet, il est inutile de créer un groupe obligatoire lorsque le segment situé au plus haut (segment de contrôle) est lui-même obligatoire. De même, il est inutile de créer un groupe facultatif lorsque le segment situé au plus haut est lui-même facultatif. Les seul cas où des structures de groupe sont indispensables sont:

1) Lorsque le groupe est facultatif et que le segment de contrôle est obligatoire.

2) Lorsque le groupe est obligatoire et que le segment de contrôle est facultatif.

Le groupe n°1 de la figure 1-7 fait bien partie de ces cas. Sans la structure de groupe, il est tout simplement impossible d'enlever l'ambiguïté entre le caractère facultatif de l'ensemble des segments DDD - EEE - FFF - GGG, et le caractère obligatoire du segment de contrôle DDD lorsqu'un des segments EEE, FFF et GGG est présent.

Les groupes peuvent aussi être imbriqués comme le montrent les segments III, JJJ et KKK de la figure 1-7. Pour le codage de ce cas-ci, après chaque apparition du segment III peuvent suivre les segments du groupe n°3 de 0 à 5 fois.

Le codage d'un message se fait dans l'ordre d'apparition des segments, c'est-à-dire de gauche à droite et en profondeur d'abord. Pour prendre un exemple concret de notre figure 1-7, imaginons un message codé dans lequel les groupes n°1 et n°3 apparaissent deux fois, et tous les autres groupes et segments apparaissent exactement une fois. L'ordre d'apparition des segments dans le codage serait alors:

UNH,AAA,BBB,CNT,MAR,DDD,EEE,FFF,GGG,DDD,EEE,FFF,GGG,HHH,III,III,KKK,III,KKK,LLL,UNT

Les règles de codage d'un message seront abordées dans 1.3.3.

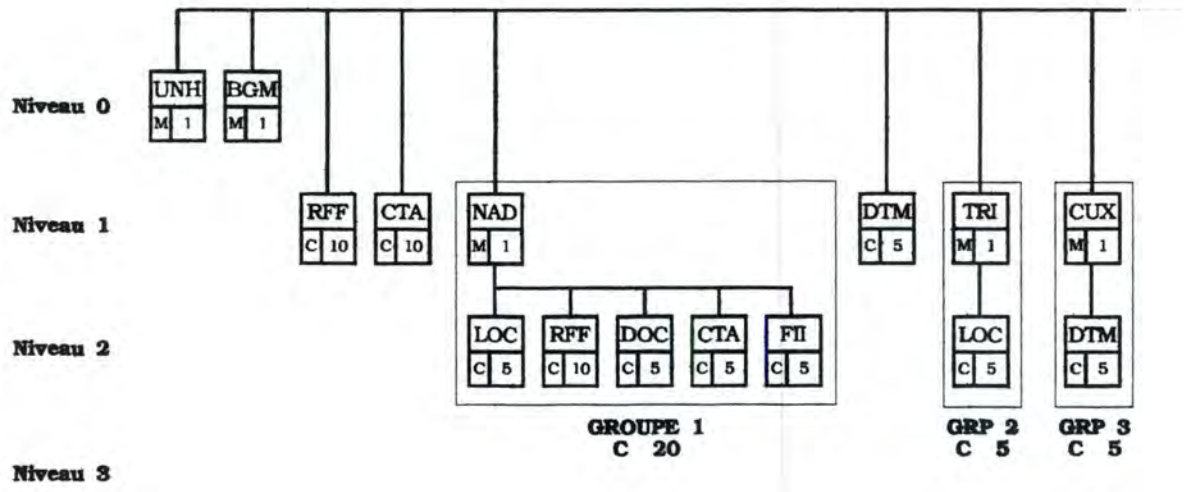

Figure 1-9 : Début du diagramme de branchement du message INVOIC

Les messages EDIFACT peuvent donc être définis à l'aide des diagrammes de branchement. En réalité, des messages tels que la facture ou le bon de commande ont des diagrammes beaucoup plus complexes que celui de notre message fictif. Nous montrons dans la figure 1-9 le début du diagramme de branchement du message INVOIC (facture); les lecteurs intéressés peuvent trouver dans [UNTDID] le diagramme complet.

1.3.2 Table de segments

La table de segments est une autre manière de définir un message. Il n'y a pas de différence fondamentale avec la définition par diagramme de branchement, si ce n'est que la présentation est ici textuelle et non graphique.

Nous donnons ici le début de la table de segments du message INVOIC, qui est l'équivalent du diagramme de branchement de la figure 1-9:

```
UNH  Message header              M  1
BGM  Beginning of message        M  1
RFF  References                  C  10
CTA  Contact                     C  10

-- Segment group 1               C  20
NAD  Name and address            M  1
LOC  Location identification     C  5
RFF  References                  C  10
DOC  Documents required          C  5
CTA  Contact                     C  5
```

```
FII  Financial institution infoC  5
DTM  Date/time reference       C  5
--   Segment group 2           C  5
TRI  Tax related information   M  1
LOC  Location identification   C  5
-- Segment group 3            C  5
CUX  Currencies                M  1
DTM  Date/time reference       C  5
.
.
.
```

Chaque ligne correspond donc à un rectangle du diagramme de branchement, c'est-à-dire à un segment avec des précisions sur le TAG, le statut (C ou M) et le nombre de répétitions. Contrairement au diagramme de branchement, le nom du segment est également précisé; mais cela est un détail sans importance.

Le premier segment qui apparaît dans un groupe est le segment de contrôle. Il est à noter que les niveaux d'imbrication ne sont pas mis en évidence dans la table de segments, mais on voit intuitivement que les segments qui suivent immédiatement le segment de contrôle sont à un niveau inférieur. Mais comment montrer une imbrication verticale de segments sans groupe comme les segments CNT et MAR de la figure 1-7?

L'UN/ECE a laissé cette question sans réponse. Ce cas précis ne s'est présenté dans aucune définition de message standard dont nous disposons, car dans un message réel, la quasi totalité des structures d'imbrication à segment de contrôle obligatoire sont facultatives, donc nécessitant un groupement. Il semblerait donc que l'UN/ECE a pensé que ce genre de situation n'arrivera pas dans un message standard. Cette lacune de la table de segments est pourtant facile à pallier: dans le cas des segments CNT et MAR, il suffirait en effet de faire apparaître la ligne du segment MAR immédiatement après celle du segment CNT, mais légèrement décalée à droite. Cela résoudrait alors le problème. Il reste à savoir si l'UN/ECE juge cette modification nécessaire, car nous le répétons, le cas ne s'est pas encore présenté dans un message réel à notre connaissance.

Si la table de segments a le mérite d'être plus facile à rédiger, le diagramme de branchement montre en revanche de manière plus nette les imbrications des segments et la structure générale du message.

1.3.3 Règles de codage du message

Le codage d'un message est une suite de segments codés selon les règles spécifiées dans 1.2.5. L'ordre d'apparition des segments dans le codage doit suivre les règles spécifiées dans 1.3.1, c'est-à-dire de gauche à droite (diagramme de branchement) ou de haut en bas (table de segments), et en profondeur d'abord. Les segments ne sont

séparés par aucun caractère de séparation, si ce n'est le caractère de fin de segment ' (apostrophe). Lorsqu'un segment facultatif est absent, il est simplement omis dans le codage et même son TAG ne doit pas y apparaître.

Les règles de codage d'un message sont donc très simples. Nous donnons ici un exemple extrait de [UNTDID] du codage d'un interchange EDIFACT contenant un message INVOIC (Note: l'exemple original de [UNTDID] comporte des erreurs et a été corrigé ici). Il est à remarquer que les coupures de ligne dans ce codage sont dues à la largeur du papier et ne sont pas présentes dans le codage original:

```
UNA:+.? 'UNB+UNOA:1+5012345678901:14+123456:91+901111:1236+R
EF01+PASSW+INVOIC'UNH+INV001+INVOIC:90:1:UN'BGM+380+75-064-H
-227101+901111'NAD+SU+5013456000145:14++ICI CHEMICALS AND PO
LYMERS+PO BOX 90:WILTON+MIDDLESBOROUGH++T56 8JE+GB'RFF+SS+ED
S0633096'RFF+PO+ABC-1234'CTA+IC++512345:TL'FII+RB+123-4567+:
::WESTLAND BANK:FRANKFURT'NAD+BY++ALPHONSO SCHMIDT AG:AVE IN
FANTI SANTO:LISBON 4:PORTUGAL'RFF+CR+064-5787-1B'NAD+CN+++QU
IMIGAL DE OPPORTO+AVE SANCHO 3+BARREIRO+++PT'CUX+DEM:IN'ALI+
GB'PAT+01+++05:03:1:60++++PAYMENT 60 DAYS FROM INVOICE DATE
BY TELEGRAPHIC TRANSFER TO:ACCOUNT NO 123-4566 QUOTE REF ABC
-1234:WESTLANDBANK,FRANKFURT'PAI+++42+03'TDT+++10+++::BAILEY
 FREIGHT'LOC+5+::TEESIDE'LOC+8+::BARREIRO'TOD+02++FRC:22+21:
::BARREIRO++TAXES AND CLEARANCE UNPAID'PAC+1++::CONTAINER'ME
A+PD+04+KG:18440'PCI++TEMP 20-25 DEG C:ALPHONSO SCHMIDT AG:0
64-5787-1B'UNS+D'LIN+++5013456000158:VN++12:18440:KG+2.85:NW
:1:KG'UNS+S'TMA+52554'FTX+CUS+++WE HEREBY CERTIFY THAT THE G
OODS MENTIONED IN THIS INVOICE ARE OF BRITISH ORIGIN'VAL+IN+
55735:DEM'UNT+28+INV001'UNZ+1+REF01'
```

Bien que nous sachions qu'il s'agit d'une suite de segments codés, il faut avouer que c'est très difficile à déchiffrer par un être humain. Pour analyser ce codage, il faut d'abord séparer les segments, puis les comparer avec leur définition dans EDSD pour comprendre les éléments de donnée qui y sont contenus. Voici le début de ce codage présenté segment par segment:

```
UNA:+.? '

UNB+UNOA:1+5012345678901:14+123456:91+901111:1236+REF01+PASS
W+INVOIC'

UNH+INV001+INVOIC:90:1:UN'

BGM+380+75-064-H-227101+901111'

NAD+SU+5013456000145:14++ICI CHEMICALS AND POLYMERS+PO BOX 9
0:WILTON+MIDDLESBOROUGH++T56 8JE+GB'

RFF+SS+EDS0633096'

RFF+PO+ABC-1234'

CTA+IC++512345:TL'

FII+RB+123-4567+::WESTLAND BANK:FRANKFURT'
```

```
NAD+BY++ALPHONSO SCHMIDT AG:AVE INFANTI SANTO:LISBON 4:PORTU
GAL'

RFF+CR+064-5787-1B'

NAD+CN+++QUIMIGAL DE OPPORTO+AVE SANCHO 3+BARREIRO+++PT'

CUX+DEM:IN'

ALI+GB'

PAT+01+++05:03:1:60++++PAYMENT 60 DAYS FROM INVOICE DATE BY
TELEGRAPHIC TRANSFER TO:ACCOUNT NO 123-4566 QUOTE REF ABC-1
234:WESTLANDBANK, FRANKFURT'

PAI+++42+03'

TDT+++10+++::BAILEY FREIGHT'

LOC+5+::TEESIDE'

LOC+8+::BARREIRO'

TOD+02++FRC:22+21:::BARREIRO++TAXES AND CLEARANCE UNPAID'

PAC+1++::CONTAINER'

MEA+PD+04+KG:18440'

PCI++TEMP 20-25 DEG C:ALPHONSO SCHMIDT AG:064-5787-1B'

UNS+D'

LIN+++5013456000158:VN++12:18440:KG+2.85:NW:1:KG'

UNS+S'

TMA+52554'

FTX+CUS+++WE HEREBY CERTIFY THAT THE GOODS MENTIONED IN THIS
 INVOICE ARE OF BRITISH ORIGIN'

VAL+IN+55735:DEM'

UNT+28+INV001'

UNZ+1+REF01'
```

Il y a d'abord le segment de service UNA, qui précise les caractères de service utilisés. Comme nous l'avons déjà dit, c'est le seul segment où il n'y a pas de caractère de séparation "+" entre le TAG et le premier élément de donnée. Ensuite, il y a le segment de service UNB, qui marque le début de l'interchange. Le segment de service UNG n'est pas présent, il n'y a donc pas de groupe fonctionnel dans cet interchange.

Le segment de service UNH marque le début du message. Puis vient le segment BGM, mais un coup d'oeil à la définition du message INVOIC nous apprend que les segments facultatifs RFF et CTA sont absents. Leur absence n'est pas signalée dans le codage, les segments sont simplement omis et le prochain segment, le segment NAD, suit immédiatement le segment BGM. Le reste suit un raisonnement similaire et ne nécessite plus de commentaire.

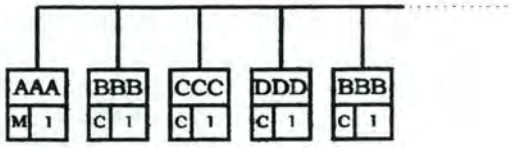

Figure 1-10 : Diagramme de branchement ambigu pour le codage EDIFACT

Une question peut venir à l'esprit: n'y a-t-il pas un risque d'ambiguïté dans cette manière de coder? Lorsqu'un segment est présent à plusieurs endroits du message, n'y a-t-il un risque de confusion? D'autant que cette crainte est justifiée par le diagramme de la figure 1-10.

Dans le codage de ce diagramme, lorsque le segment BBB apparaît après le segment AAA, nous nous trouvons dans l'impossibilité de déterminer de quel segment BBB il s'agit, puisque les segments CCC et DDD sont facultatifs et sont susceptibles d'être omis dans le codage. Il y a donc un risque d'ambiguïté, d'autant que la figure 1-10 ne présente pas le seul cas de confusion possible et que beaucoup d'autres situations sont jugées ambiguës comme l'a montré [BLOCH].

Heureusement, si le danger est bien réel en théorie, il ne l'est plus en pratique. En effet, les responsables de l'UN/ECE sont conscients du problème et ils disposent des algorithmes pour détecter les ambiguïtés lors de la définition d'un message. Dès lors, on peut estimer que les messages standard EDIFACT sont corrects et sans ambiguïté.

Chapitre 2

Introduction à la Recommandation X.400

La Recommandation X.400 décrit le service de messagerie électronique défini par CCITT. Elle a été ratifiée comme étant un standard international par l'ISO sous le nom de MOTIS (*Message-Oriented Text Interchange System*). La version originale de X.400 a été définie en 1984, une version améliorée comportant des services additionnels a ensuite été définie en 1988. Il existe donc deux versions de X.400: la version 1984 et la version 1988.

Note: Les abréviations suivantes seront utilisées tout au long de ce texte:

X.400-84 = La Recommandation X.400, version 1984
X.400-88 = La Recommandation X.400, version 1988

La Recommandation X.400 est en réalité constituée de toute une série de documents dont chacun décrit un aspect bien précis de X.400. La table 2-1 montre la liste de ces documents et leur référence dans les deux versions de X.400.

1984	1988	Sujet
X.400	X.400	Vue générale du système et éléments de service
X.401		Eléments de service de base et facilités optionnelles d'utilisateur
	X.402	Architecture générale
	X.403	Test de conformité
	X.407	Conventions de définition de service abstrait
X.408	X.408	Règles de conversion de type d'information encodée (EIT)
X.409	X.208(*)	Notation de syntaxe de présentation (Langage ASN.1)
	X.209(*)	
X.410		*"Remote operations" et "Reliable transfer server"*
X.411	X.411	Système de transfert de message: services abstraits et procédures
	X.413	Message Store: services abstraits
	X.419	Spécifications de protocoles (Protocoles P1, P3 et P7)
X.420	X.420	Système de messagerie interpersonnelle
X.430		Protocoles d'accès pour terminaux Télétex

(*) Ne fait pas partie de X.400-88

Table 2-1 : Documents de la Recommandation X.400

Nous allons d'abord décrire les aspects de X.400 qui sont communs à X.400-84 et X.400-88 (paragraphes 2.1 à 2.4), avec éventuellement des explications sur les différences qui existent entre les deux versions. Les nouveaux concepts introduits dans X.400-88 seront discutés à la fin de ce chapitre (paragraphe 2.5).

2.1 Architecture générale de X.400

X.400 est basé sur deux composants principaux: le MTA (*Message Transfer Agent*) et l'UA (*User Agent*). Le MTA est responsable du transfert de message entre l'émetteur et le récepteur. L'UA est une interface qui interagit directement avec l'utilisateur et fournit à celui-ci les fonctions qui lui permettent d'envoyer et de recevoir des messages.

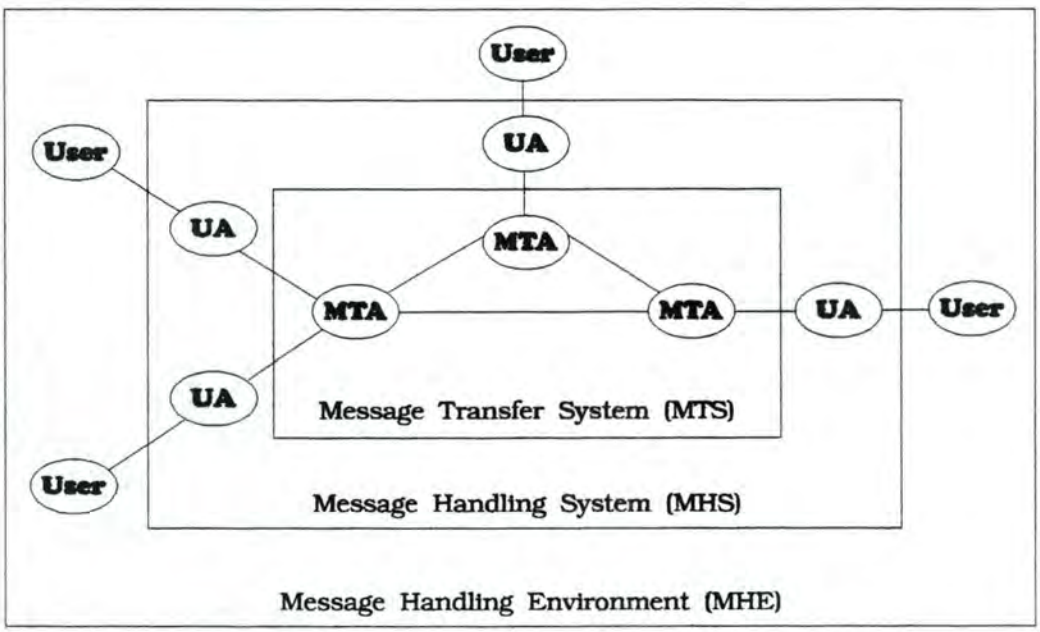

Figure 2-1 : Modèle fonctionnel du service de messagerie X.400

La figure 2-1 montre le modèle fonctionnel de X.400. On peut diviser le service de messagerie X.400 en trois couches:

-MTS: Les MTAs forment le MTS (*Message Transfer System*), ou Système de Transfert de Message. Le protocole utilisé entre les MTAs est appelé le protocole P1.

-MHS: Les UAs et le MTS forment le MHS (*Message Handling System*), ou Système de Traitement de Message. Dans un système de messagerie interpersonnelle (voir 2.3), le protocole utilisé entre les UAs est appelé le protocole P2.

-MHE: Les utilisateurs et le MHS forment le MHE (*Message Handling Environment*), ou Environnement de Traitement de Message. Le protocole utilisé entre les utilisateurs est bien sûr leur protocole privé et ne fait pas partie de X.400.

X.400 fonctionne selon le principe de " *store-and-forward*". Cela signifie que l'établissement préalable d'une connexion entre l'émetteur et le récepteur n'est pas nécessaire et que le transfert d'un message se fait par les étapes suivantes:

1) Préparation du message

L'UA de l'émetteur assiste celui-ci dans la création et la composition du message.

2) Soumission du message

L'UA de l'émetteur soumet le message à son MTA et lui fournit les informations nécessaires à la création de l'enveloppe.

3) Relais du message

Le MTA se charge du problème de routage en se basant sur les informations de l'enveloppe. Le message accompagné de son enveloppe passe de MTA à MTA pour arriver au MTA destinataire (le MTA qui est connecté à l'UA du récepteur).

4) Livraison du message

Le MTA destinataire délivre le message à l'UA du récepteur si celui est en état de fonctionnement. Si au contraire l'UA du récepteur est momentanément coupé comme cela peut arriver dans une petite entreprise, alors le MTA destinataire garde le message pour le délivrer plus tard, pour autant que sa capacité de stockage soit suffisante.

5) Réception du message

L'UA du récepteur confie le message au récepteur.

Le principe "*store-and-forward*", qui permet de délivrer un message "au moment opportun", a le mérite d'être très souple et convient très bien à un service de messagerie électronique. De plus, comme le routage dans X.400 est fait par le MTA - donc au niveau Application, X.400 est très indépendant des équipements de télécommunication et des protocoles de bas niveau. [GENILLOUD]

Il est utile de donner quelques explications ici sur le mot "message" pour éviter toute confusion. Le message qui circule dans le MTS est en réalité une enveloppe accompagnée d'un contenu, c'est-à-dire d'un message de n'importe quel type. L'enveloppe et le contenu sont régis par le protocole P1 et sont appelés le **message**

P1. Quant au contenu - c'est-à-dire le message échangé entre les UAs, CCITT n'en a défini qu'un seul type dans X.400: le message interpersonnel (*Interpersonal Message* ou IPM). Ce message est régi par le protocole P2 et est appelé le **message P2**. (Voir figure 2-2)

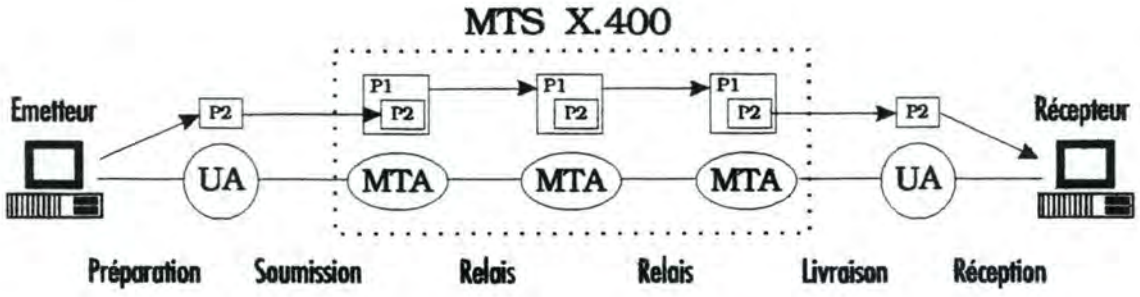

Figure 2-2 : Transfert de message interpersonnel par le service de messagerie X.400

L'UA discuté dans ce chapitre est donc un UA de messagerie interpersonnelle. Il peut exister d'autres types d'UA comme nous le verrons dans le chapitre 3.

2.2 Système de transfert de message (MTS)

2.2.1 Types de message P1

Le MTS est l'ossature de base du service de messagerie X.400. Il est chargé de convoyer des messages de tout type de l'émetteur au récepteur. Trois types de messages P1 sont utilisés dans le MTS:

Message-utilisateur (*User Message*):
Le type de message de base, utilisé pour l'échange de données entre deux utilisateurs.

Probe:
"Sonde" qui permet à un émetteur de tester la présence du récepteur, ainsi que la capacité de ce dernier de recevoir des messages de certains types et de certaines longueurs. Le *probe* évite à l'émetteur d'envoyer inutilement un message à un récepteur dont l'adresse est incorrecte ou qui n'est pas capable de le recevoir.

Rapport de livraison (*Delivery Report*):
Type de message généré par le MTS pour informer l'émetteur du succès ou non de la livraison du message-utilisateur ou du *probe* envoyé précédemment.

La structures de ces trois types de message est montrée dans la figure 2-3.

Le message-utilisateur consiste en une enveloppe ainsi que le message échangé entre

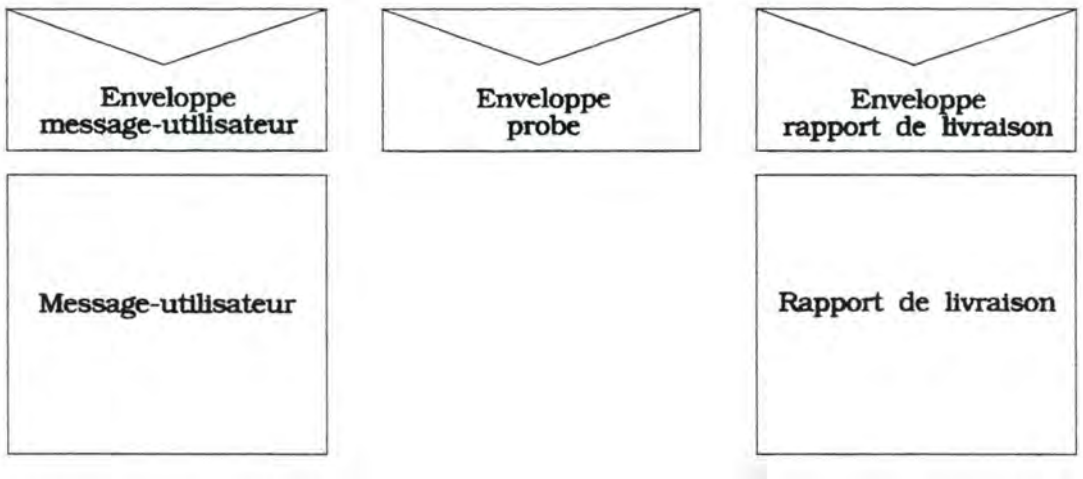

Figure 2-3 : Structure de base des trois types de message P1

deux utilisateurs. Sur l'enveloppe figurent les champs de données requises par le MTS pour l'envoi au récepteur. Les plus importants de ces champs de données sont:

-MPDUIdentifier: L'identifiant du contenu. (Généré par l'UA pour la corrélation avec un éventuel rapport de livraison)

-Originator: L'*O/R Name* de l'émetteur. (Voir 2.5.1 pour *O/R Name*)

-RecipientInfo: L'*O/R Name*(s) du(des) récepteur(s), et aussi la demande de rapport(s) de livraison (permettre au MTS de savoir s'il faut générer un rapport de livraison/non-livraison).

-ContentType: Le type de contenu.

-OriginalEncodedInformationTypes: Les types d'information encodée (EITs). (Fac-similé, télex, ...)

-Priority: La priorité du message.

X.400-88 introduit d'autres champs de données pour l'enveloppe du message-utilisateur. Ce sont principalement des champs qui concernent la sécurité (voir 2.6.4), notamment le champ *Content-integrity-check* dont nous verrons l'usage dans 4.3.2.

Le *probe* consiste en une enveloppe vide, sans aucun contenu. La structure de l'enveloppe du probe est quasi identique à celle du message-utilisateur, excepté un champ de donnée additionnel qui indique la longueur du message que l'émetteur a l'intention d'envoyer. Lorsqu'un émetteur est prêt à envoyer un message (P2 ou autres), il peut d'abord envoyer au récepteur un *probe* qui est en réalité l'enveloppe du message-utilisateur qui contiendra le message à envoyer. Selon le type du rapport reçu (livraison ou non-livraison), l'émetteur saura s'il peut envoyer le message. Le *probe* peut être très utile lorsque le message à envoyer est très volumineux et qu'on ne souhaite pas prendre le risque de gaspiller les ressources de télécommunication en

l'envoyant inutilement.

Le rapport de livraison a une structure assez différente de celle du message-utilisateur et du *probe*. L'enveloppe du rapport contient principalement l' *O/R Name* de l'émetteur qui a provoqué la génération du rapport, autrement dit l'adresse à laquelle le MTS doit envoyer le rapport. Le contenu du rapport comporte principalement les champs de données suivants:

-Identifiant du contenu. (Celui qui se trouvait sur l'enveloppe du message-utilisateur ou du *probe* qui a provoqué la génération du rapport)

-Résultat du rapport. (Livraison ou non-livraison)

-Raison de la non-livraison.

X.400-88 introduit d'autres champs de données, principalement ceux concernant la sécurité (voir 2.6.4).

Par défaut, l'émetteur reçoit un rapport seulement en cas de non-livraison. Mais il peut très bien demander au MTS de ne jamais lui envoyer de rapport ou de lui envoyer un rapport dans tous les cas.

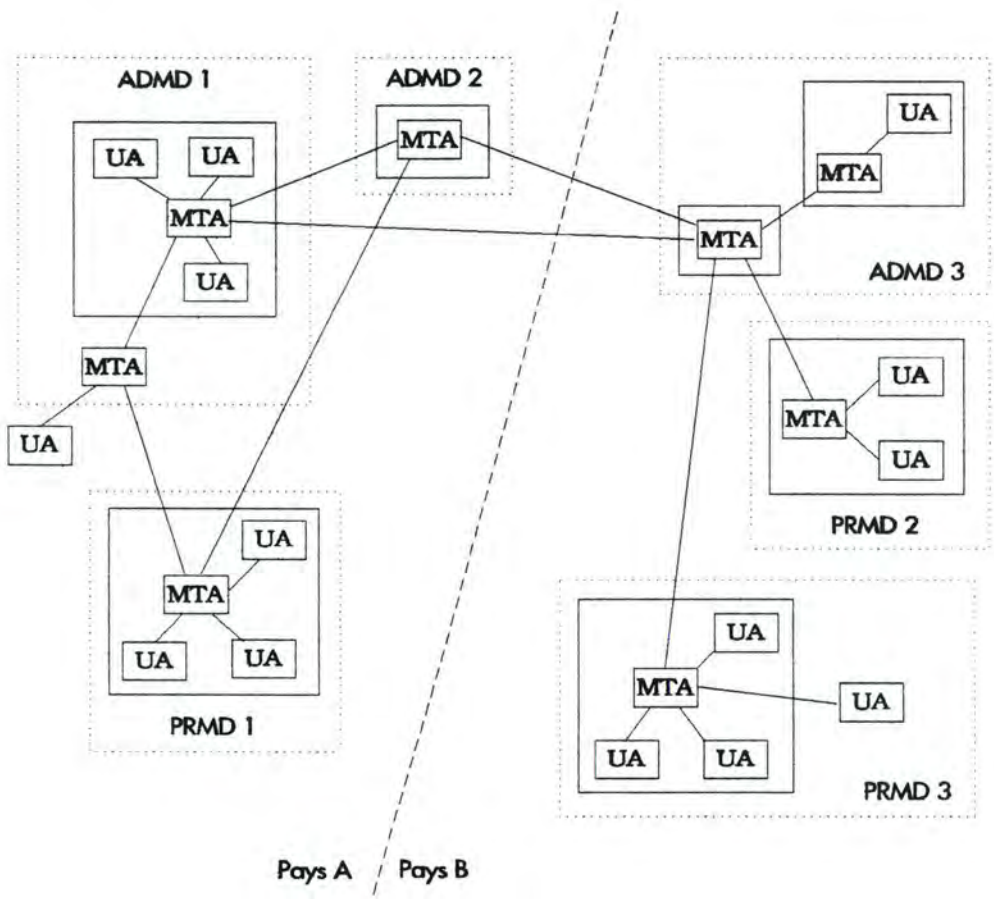

Figure 2-4 : ADMDs et PRMDs

2.2.2 *Management domain* [HS]

Le MTS est partitionné en un nombre de *management domains* (MDs). Chaque MD contient au moins un MTA et zéro ou plus d'UAs. L'objectif de la division du MTS en domaines est de déléguer la responsabilité de l'organisation et de la gestion du système aux autorités capables d'assumer cette tâche. Chaque domaine est responsable de sa gestion interne, en particulier du routage approprié des messages et de leur livraison. Un message qui entre dans un domaine reste sous sa responsabilité jusqu'au moment où il est délivré dans le domaine ou relayé à un domaine externe.

Un MD géré par une administration CCITT (ex.: un PTT) est appelé un *Administration Management Domain* (ADMD), et un MD géré par une organisation privée est appelé un *Private Management Domain* (PRMD). Un PRMD doit résider à l'intérieur d'un pays et il doit être associé à au moins un ADMD de son pays. (Voir figure 2-4)

Un rôle important de l'ADMD est d'agir en tant qu'autorité de nomination (*naming authority*) pour tous les PRMDs qui sont sous sa responsabilité. Cela signifie que l'ADMD supervise et arbitre l'attribution des noms aux PRMDs. L'ADMD est concerné seulement par l'administration des noms de haut niveau; la responsabilité pour l'attribution des noms à l'intérieur du PRMD est normalement déléguée à celui-ci.

2.2.3 Services de transfert de messages

Le MTS fournit des **services** de transfert de messages (*MT services*) à l'utilisateur du MTS afin de lui permettre d'accéder au MTS pour échanger des messages. Ces services sont basés sur des **éléments de service** de transfert de messages (*MT service elements)* décrits dans 4.1 de X.400-84 ou dans 8 et 19 de X.400-88.

Il faut bien faire la distinction entre le service et l'élément de service. Le service est une opération offerte par le système; dans un MTS, le service permet par exemple à l'UA de soumettre ou de recevoir des messages. L'élément de service est une fonction ou une capacité particulière du système sur lequel se basent des services; dans un MTS, l'élément de service permet par exemple de préciser la priorité du message, de demander un rapport de livraison, etc.

Les éléments de service MT sont divisés en cinq groupes:

1) **Basic MT service elements**

   Eléments de service qui permettent à l'UA d'envoyer et de recevoir des messages P1, d'établir une association avec le MTS, de recevoir un rapport de non-livraison, d'indiquer au MTS les types d'information encodée (EITs) du message soumis, etc. Ce groupe est le seul qui soit non-optionnel.

2) **Submission and delivery service elements**

Eléments de service qui permettent à l'UA d'indiquer la priorité du message, de soumettre un message à plusieurs récepteurs, de différer la livraison d'un message, de demander un rapport de livraison, de demander de ne pas recevoir un rapport de non-livraison, etc.

3) **Conversion service elements**

Eléments de service qui permettent à l'UA de l'émetteur de demander la conversion implicite des types d'information encodée (EITs), de demander la conversion explicite des EITs avec indication au récepteur, et d'interdire la conversion des EITs. Ces éléments de service sont très utiles lorsque l'émetteur et le récepteur n'utilisent pas les mêmes EITs.

4) **Query service elements**

Eléments de service qui permettent à l'UA de soumettre un *probe.*

5) **Status and inform service elements**

Eléments de service qui permettent à un UA de demander la livraison des messages dont l'adresse du destinataire ne correspond pas exactement à la sienne, et de demander au MTS de garder momentanément les messages qui lui sont destinés pour les délivrer plus tard.

X.400-88 offrent d'autres éléments de services, principalement des éléments de service de *Message Store*, de sécurité et de PDAU. Ils seront discutés dans 2.6 et 4.5.

2.3 Système de messagerie interpersonnelle

Le système de messagerie interpersonnelle (IPMS) est formé du MTS et des UAs conçus pour le traitement des messages interpersonnels. Il fournit des services à un individu et assiste ce dernier pour communiquer avec d'autres individus. L'IPMS est donc un cas particulier du MHS.

2.3.1 Message interpersonnel

Le message échangé dans un IPMS est un message interpersonnel (IPM) ou message P2. L'IPM est divisé en deux parties: l'en-tête (*heading*) et le corps (*body*). Le *body* consiste en une séquence de *body parts.* Un *body part* est en général un texte compréhensible par l'être humain, mais peut aussi être du fac-similé, de la voix, du télex, des dessins, ... etc. Ainsi, contrairement à ce qu'on aurait pu penser, un message interpersonnel ne contient pas que du texte. (Voir figure 2-5)

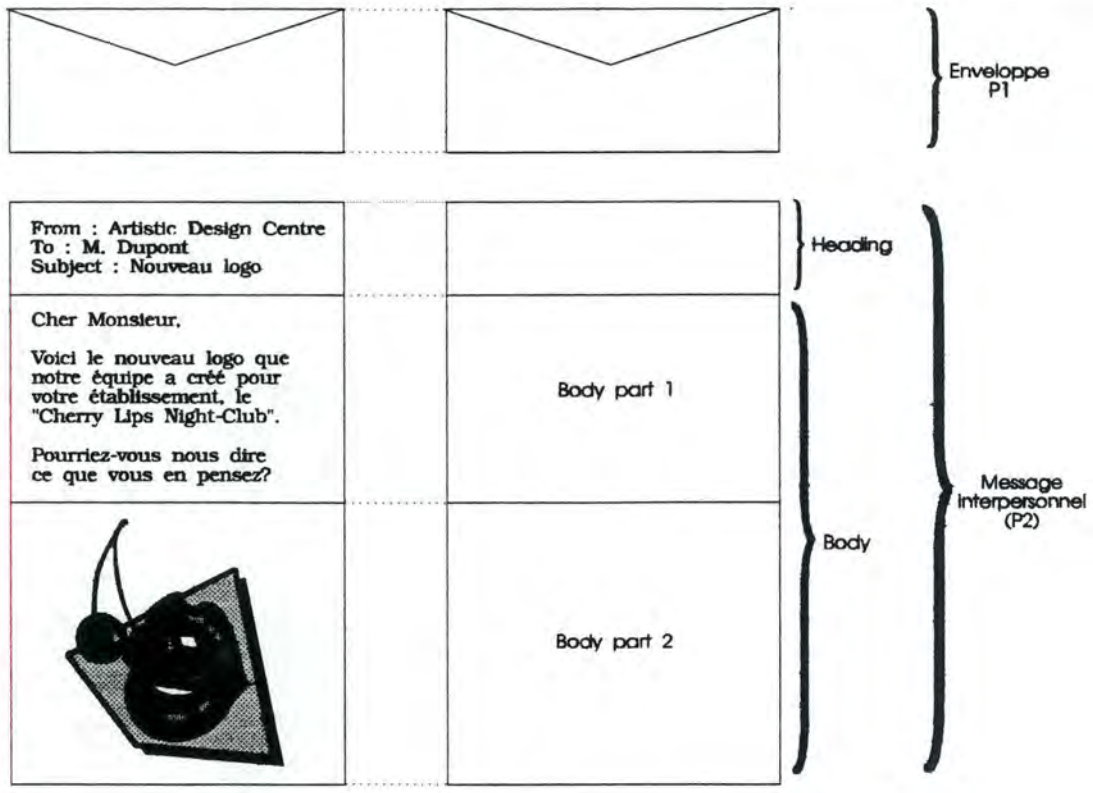

Figure 2-5 : Exemple d'un message interpersonnel

Le *heading* comporte des champs de données qui contiennent des informations relatives au *body*. Voici une brève description des plus importants de ces champs:

-IPMessageIP: L'identifiant du message interpersonnel.

-Originator: L'*O/R Name*, le nom de forme libre (ex.: ''Jean-Pierre'') ou le numéro de téléphone de celui qui a soumis l'IPM.

-PrimaryRecipients: Les *O/R Names*, les noms de forme libre ou les numéros de téléphone des récepteurs principaux, avec la demande éventuelle de notification. (Voir 2.3.2 pour la notification interpersonnelle)

-CopyRecipients (CC): Les *O/R Names*, les noms de forme libre ou les numéros de téléphone des récepteurs qui reçoivent une ''copie-carbone'' de l'IPM, avec la demande éventuelle de notification.

-BlindCopyRecipients (BCC): Les *O/R Names*, les noms de forme libre ou les numéros de téléphone des récepteurs secrets qui reçoivent une ''copie-carbone'' de l'IPM, avec la demande éventuelle de notification. Les ''BlindCopyRecipients'' reçoivent le message sans que les autres récepteurs, qu'ils soient ''PrimaryRecipients'' ou ''CopyRecipients'', ne soient au courant.

-InReplyTo: L'identifiant de l'IPM auquel cet IPM répond.

-Obsoletes: Les identifiants des IPMs qui sont rendus périmés par cet IPM.

-Subject: Le sujet de l'IPM.

-ExpiryDate: La date d'expiration de l'IPM.

-Importance: L'importance de l'IPM. Trois valeurs sont possibles: *low, normal, high.*

-Sensitivity: La sensibilité de l'IPM. Trois valeurs sont possibles: *personal, private, company confidential.*

-Autoforwarded: Champ indiquant si l'IPM a été relayé automatiquement.

Quelques explications sur ce dernier champ sont nécessaires. Lors de la réception d'un IPM, le récepteur peut décider de le relayer à un autre utilisateur. Cependant, un utilisateur peut aussi demander à son UA de relayer automatiquement certains types d'IPM. Cette dernière action s'appelle le relais automatique ou *auto-forward.*

2.3.2 Notification interpersonnelle

La notification interpersonnelle (IPN) est un message généré par l'UA récepteur pour informer l'UA émetteur de la bonne réception ou non de l'IPM. La génération de l'IPN dépend des actions prises par l'émetteur d'un IPM: celui-ci pouvait en effet demander le renvoi d'une IPN de réception et/ou de non-réception à chaque récepteur (qu'il soit récepteur principal, CC ou BCC) lors de l'envoi de l'IPM.

Il est très important de ne pas confondre la notion de l'IPN avec la notion du rapport de livraison. Le rapport de livraison confirme que le message a successivement traversé le MTS et est arrivé à l'UA du récepteur, alors que l'IPN confirme que le message a bien été reçu par le récepteur lui-même. Autrement dit, le rapport de livraison est une confirmation de MTA à MTA dans le MTS, alors que l'IPN est une confirmation d'UA à UA dans l'IPMS. L'IPN est mise dans un **message-utilisateur** (et non dans un rapport) pour son renvoi à l'émetteur de l'IPM.

Il y a deux sortes d'IPN: l'IPN de réception et l'IPN de non-réception. Elles comportent essentiellement les champs de donnée suivants:

Champs communs aux deux IPNs:

-Reported: L'identifiant du message interpersonnel qui a provoqué la génération de l'IPN.

-ActualRecipient: L'*O/R Name*, le nom de forme libre ou le numéro de téléphone de l'émetteur de l'IPN (= le récepteur de l'IPM).

Champs de l'IPN de réception:

-Receipt: La date et l'heure de la réception de l'IPM.

-TypeOfReceipt: Le type de l'IPN de réception. Deux valeurs sont possibles: ''explicit'' signifie que l'IPN est générée suite à la demande du récepteur, et ''automatic'' signifie que l'IPN est générée automatiquement par l'UA du récepteur.

Champs de l'IPN de non-réception :

-Reason: La raison de la non-réception. Deux valeurs sont possibles: ''UAEInitiatedDiscard'' signifie que l'UA a refusé l'IPM pour des raisons locales, et ''Autoforwarded'' signifie que l'UA a relayé l'IPM à un autre UA.

-NonReceiptQualifier: L'indication du refus de l'IPM, utilisé lorsque Reason=''UAEInitiatedDiscard''. Trois valeurs sont possibles: ''expired'' signifie que l'IPM est expiré lors de sa réception, ''obsoleted'' signifie que l'IPM est rendu périmé par un autre IPM, et ''SubscriptionTerminated'' signifie que l'abonnement de l'émetteur de l'IPM est terminé.

-Comments: Commentaire du récepteur lorsque Reason=''Autoforwarded''.

2.3.3 Services de messagerie interpersonnelle

L'IPMS fournit des services de messagerie interpersonnelle (*IPM services*) qui permettent à un individu d'envoyer et de recevoir des IPMs. Les services de messagerie interpersonnelle sont basés sur les éléments de service MT (voir 2.2.3) et ceux spécifiques à l'IPMS. L'ensemble de ces éléments de services forme les éléments de service de messagerie interpersonnelle (*IPM service elements*).

Les éléments de service IPM sont divisés en six groupes:

1) **Basic IPM service elements**

Eléments de service qui permettent à l'utilisateur de l'IPMS d'envoyer et de recevoir des IPMs, d'attribuer un identifiant aux IPMs, et d'inclure différents types de *body part* dans un IPM. Ce groupe inclut les éléments de service de base MT (voir groupe 1 dans 2.2.3).

2) **Submission and delivery, and conversion service elements**

(Voir groupes 2 et 3 dans 2.2.3)

3) **Cooperating IPM UA action service elements**

Eléments de service qui permettent à l'utilisateur de l'IPMS de spécifier des récepteurs de *Blind Copy* (BCC, voir 2.3.1), de demander une IPN de non-réception, d'être avertis de la réception des IPNs, et de déterminer si l'IPM reçu

a été relayé automatiquement (*auto-forwarded*).

4) **Cooperating IPM UA information conveying service elements**

Eléments de service qui permettent à l'utilisateur de l'IPMS d'indiquer l'identifiant de l'émetteur, d'indiquer les identifiants des récepteurs principaux et des récepteurs de *Copy* (CC, voir 2.3.1), d'indiquer la date d'expiration de l'IPM, d'indiquer l'importance et la sensibilité de l'IPM, d'indiquer le sujet de l'IPM, ... etc.

5) **Query service elements**

(Voir groupe 4 dans 2.2.3)

6) **Status and inform service elements**

(Voir groupe 5 dans 2.2.3)

2.4 X.400 dans la couche Application OSI

Le service de messagerie X.400 est une application du modèle OSI. Après avoir vu le modèle fonctionnel du MHS X.400, il est certainement intéressant de voir comment le MHS X.400 s'insère dans le modèle OSI.

Les fonctions du traitement de messages X.400 dans la couche Application peuvent être divisées en deux sous-couches [X.400-84]:

1) La sous-couche UA (UAL) contient la fonctionnalité de l'UA associée aux contenus des messages.

2) La sous-couche de transfert de message (MTL) contient le fonctionnalité du MTA et fournit les services MT.

L'entité UA (UAE) est la fonctionnalité qui permet à l'UA de coopérer avec d'autres UAs dans un UAL. Cette coopération se fait via un protocole dépendant du type de message traité de l'UA. Lorsque le type n'est pas précisé, ce protocole porte un nom général: le protocole Pc. Dans le cas de l'UA de messagerie interpersonnelle, c'est le protocole P2 qui est utilisé entre les UAEs.

L'entité MTA (MTAE) est la fonctionnalité qui permet au MTA de coopérer avec d'autres MTAs dans un MTL. Cette coopération se fait via le protocole P1.

Jusqu'à présent, nous avons supposé qu'un UA est toujours directement connecté à un MTA. Or, le MTA étant un composant coûteux, certains systèmes ne comportent qu'un UA connecté à un MTA extérieur, par exemple un MTA public ou un MTA privé d'un VAN. Dans ce cas-ci, pour que l'UAE puisse accéder au MTAE, il faut nécessairement passer par une entité de soumission et de livraison (SDE). Le SDE est

la fonctionnalité du MTL qui permet à l'UAE d'accéder à distance au MTAE. La coopération entre le SDE et le MTAE se fait via le protocole P3.

La figure 2-6 montre la disposition de ces différentes entités ainsi que les protocoles utilisés dans l'IPMS.

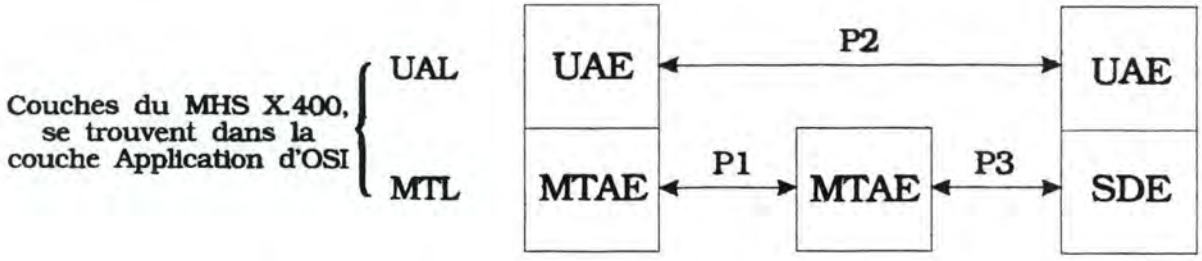

Figure 2-6 : Description "en couches" de l'IPMS

2.5 Adressage dans X.400

2.5.1 *O/R Name* et *O/R Address*

Lorsqu'un émetteur du MTS soumet un message, il doit fournir au MTS le(s) nom(s) du(des) récepteur(s). Ce nom est l' *O/R Name* (*Originator/Recipient Name*), qui **désigne un ou plusieurs utilisateurs du MTS**.

Chaque utilisateur du MTS est **identifié** par une *O/R Address* (*Originator/Recipient Address*). L' *O/R Address* est un ensemble d'attributs qui permet aux MTAs de trouver un chemin pour envoyer un message à **un utilisateur unique du MTS**.

Puisqu'un *O/R Address* permet de désigner un utilisateur particulier du MTS, un *O/R Address* est aussi un *O/R Name*. Un *O/R Name* est toujours une *O/R Address* dans X.400-84, mais il peut être soit un *O/R Address* ou un *Directory Name* (Nom de répertoire) dans X.400-88. Nous allons discuter de ce dernier cas dans 2.6.3. Pour la suite de ce paragraphe, les termes *O/R Name* et *O/R Address* ont une signification identique.

L' *O/R Name* comporte certains des attributs suivants:

-Country Name
-ADMD Name
-X.121 Address
-Telematic Terminal ID
-PRMD Name
-Organization Name
-Unique UA Identifier
-Personal Name (comprenant Surname, Given name, Initials et Generational

qualifier)
-Organizational Unit(s)
-Domain Defined Attributes

Dans X.400-88, un groupe d'utilisateurs du MTS peut être désigné par une liste de distribution (*distribution list* ou DL). La liste de distribution est un cas particulier d'*O/R Name* qui permet par exemple de désigner tous les membres d'une organisation. Dans ce cas-ci, lorsqu'on voudrait envoyer un message d'intérêt général à tout le monde, l'émetteur peut spécifier comme récepteur une liste de distribution sans devoir donner la liste complète des *O/R Names* des membres de l'organisation. Quand un MTA reçoit un message dont la destination est spécifiée par une liste de distribution, il peut effectuer une "expansion" sur le message et l'envoie à plusieurs autres MTAs (ou aux UAs directement connectés à ce MTA). Le processus se répète jusqu'à ce que tous les membres de la liste de distribution reçoivent le message.

Il faut remarquer que l' *O/R Name* d'une liste de distribution et l' *O/R Name* "normal" ont des apparences semblables et qu'un UA est incapable de savoir de quel type d' *O/R Name* il s'agit lors de la soumission d'un message.

2.5.2 Construction de l' *O/R Name*

Tous les attributs de l' *O/R Name* ne doivent pas nécessairement être utilisés pour identifier un utilisateur. X.400 définit quatre formes de construction de l' *O/R Name*. Chaque forme spécifie un ensemble d'attributs qui permet d'identifier un utilisateur unique (ou un groupe d'utilisateurs unique s'il s'agit d'une liste de distribution) dans un MTS.

Nous allons présenter ces formes de construction de l' *O/R Name*. Les attributs mis entre crochets ([]) sont des attributs optionnels:

1) Country Name
   ADMD Name
   [PRMD Name]
   [Personal Name]
   [Organization Name]
   [Organizational Unit Name]
   [Domain Defined Attributes]

Au moins un des cinq attributs optionnels doit être sélectionné.

2) Country Name
   ADMD Name
   Unique UA Identifier
   [Domain Defined Attributes]

3) Country Name
   ADMD Name

X.121 Address
[Domain Defined Attributes]

4) X.121 Address
[Telematic Terminal ID]

Country Name = BE
ADMD Name = RTT
PRMD Name = FUNDP
Organization Name = INFO
Surname = Van Bastelaer
Given Name = Philippe

Figure 2-7 : Exemple de construction d'O/R Name

La figure 2-7 montre un exemple d' *O/R Name* basé sur la forme 1 ci-dessus.

2.6 Nouveaux concepts de X.400 version 1988

Bien que la version 84 de X.400 ait apporté beaucoup d'innovations par rapport aux systèmes de messagerie existants, elle n'est certainement pas parfaite. D'abord, plusieurs concepts étaient simplement cités sans aucune explication; c'est notamment le cas de l'utilisation des services de répertoire X.500, qui est laissée "pour étude

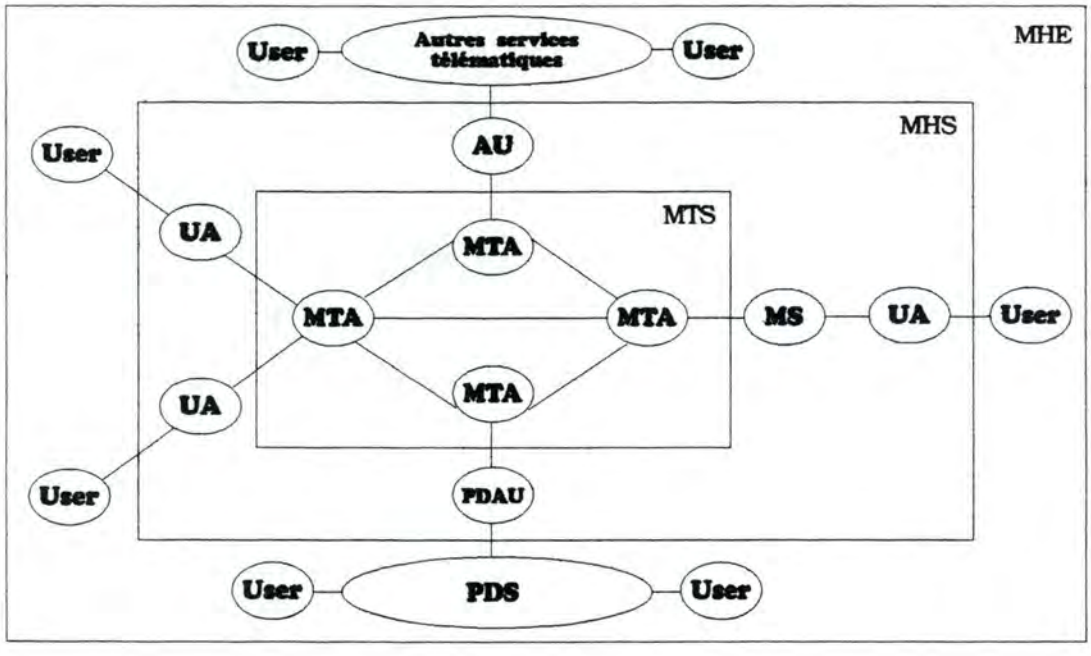

Figure 2-8 : Modèle fonctionnel du service de messagerie X.400 version 1988

ultérieure''. Ensuite, la couche Application du modèle OSI n'était pas encore complètement définie lors de l'apparition de X.400-84, ce qui rend l'intégration de X.400-84 dans l'OSI imparfaite et inachevée.

X.400-88 apporte des améliorations qui rendent le MHS X.400 plus sûr, plus polyvalent et plus souple pour ses utilisateurs. La figure 2-8 montre le nouveau modèle fonctionnel de X.400-88 (à comparer avec la figure 2-1). De nouveaux composants apparaissent dans la figure:

-MS: *Message Store*
-AU: *Access Unit*
-PDAU: *Physical Delivery Access Unit*
-PDS: *Physical Delivery Services*

Nous allons décrire ces nouveaux composants, ainsi que d'autres nouveaux concepts introduits dans X.400-88.

2.6.1 *Message Store*

Puisque X.400 est une application d'OSI, il est censé pouvoir fonctionner indépendamment des systèmes utilisés. Ainsi, il pourrait très bien tourner sur des PCs, genre de machine de plus en plus utilisée aujourd'hui. Nous savons également que dans une petite entreprise, les PCs ne fonctionnent pas toujours en permanence et ils peuvent très bien être coupés assez fréquemment. Se pose alors le problème de la coupure d'un PC sur lequel tourne un UA.

Lorsque l'UA est hors fonctionnement, le MTA ne peut bien entendu lui délivrer les messages qui lui sont destinés. Dans ce cas-ci, le MTA est normalement capable de garder momentanément les messages en attendant que l'UA soit de nouveau en fonctionnement. Mais deux problèmes risquent néanmoins d'arriver:

-Si la quantité de messages en attente est trop importante, l'UA risque d'être submergé de messages une fois sa connexion avec le MTA rétablie. En effet, les PCs ont en général une capacité de stockage assez limitée et une arrivée soudaine et trop importante de messages peut causer la perte de ces derniers.

-Si l'UA est coupé pour trop longtemps - ne fût-ce une période de 24 heures, le MTA peut déduire que l'UA est en panne et renvoyer un rapport de non-livraison aux émetteurs des messages. Le récepteur risque ainsi de perdre des messages que pourtant il souhaite recevoir.

Pour ces raisons, CCITT a introduit dans X.400-88 (la Recommandation X.413) une entité fonctionnelle qui est placée entre le MTA et l'UA: le *Message Store* (MS). Le MS est un composant optionnel de X.400-88 qui possède une grande capacité de stockage, qui est en fonctionnement permanent, et qui est capable de stocker de manière fiable les messages reçus par le MTA et destinés à l'UA. Lorsque le MS est présent entre le MTA et l'UA, le MTA ne délivre pas les messages directement à l'UA mais bien au MS. Le MS stocke alors les messages et attend que l'UA vienne

les chercher. Le protocole utilisé entre le MS et l'UA s'appelle le protocole P7.

Attributs généraux du MS

Pour chaque message P1 qu'il reçoit du MTA, le MS doit créer et gérer une liste de 42 attributs. Ces attributs s'appellent les attributs généraux (*general attributes*, voir 11 et tables 1 & 2 dans X.413). La plupart de ces attributs sont construits à partir des paramètres des services abstraits MessageDelivery et ReportDelivery définis dans le paragraphe 8 de la Recommandation X.411 (autrement dit, construits à partir de l'enveloppe P1 du message), d'autres sont générés directement par le MS. Chaque entrée d'attributs est stockée dans une base de données nommée *information-base*. L'*information-base* comporte donc une séquence d'entrées d'attributs qui sont prêtes à être consultées par l'UA.

Voici quelques-uns de ces attributs généraux:

-Content-type
-Priority
-Message-delivery-time
.
.
.

Recherche sélective

Grâce aux attributs généraux, l'UA peut effectuer des recherches sélectives en se basant sur les valeurs de ces attributs. L'UA peut par exemple demander au MS de lui fournir tous les messages dont *Content-type* = P2, *Priority* = "urgent", et *Message-delivery-time* = la date du jour. Le MS consulte alors son *information-base* et fournit à l'UA les messages qui répondent à ces critères.

Auto-action

Le concept d'*auto-action* est décrit dans 6.5 et 12 de la Recommandation X.413. L'*auto-action* est une action que le MS exécute automatiquement pour le compte de l'UA. Dans X.413 sont définies deux *auto-actions* générales, elles peuvent potentiellement être utilisées pour les messages P1 contenant tout type de contenu. Elles sont:

1) Auto-forward (Relais automatique):

Le MS relaie automatiquement le message reçu si celui-ci répond aux critères préalablement définis par l'UA.

2) Auto-alert (Avertissement automatique):

Le MS avertit automatiquement l'UA de l'arrivée d'un message si celui-ci

répond aux critères préalablement définis par l'UA.

Pour chacune de ces *auto-actions*, l'UA doit fournir préalablement un ou plusieurs ensemble de valeurs d'attributs au MS comme critères. Le MS pourra alors consulter son *information-base* et effectuer une *auto-action* pour les messages P1 dont les valeurs d'attributs correspondent à celles fournies par l'UA.

Grâce aux attributs généraux sur lesquels se basent les recherches sélectives et les *auto-actions*, le MS peut devenir beaucoup plus intelligent qu'une simple unité de stockage. L'introduction du *Message Store* est certainement l'amélioration la plus importante apportée dans X.400-88.

2.6.2 *Access Unit* et *Physical Delivery Access Unit*

Le *Access Unit* (AU) et le *Physical Delivery Access Unit* (PDAU) sont des composants optionnels introduits dans X.400-88. Ces composants permettent au MHS X.400 d'établir des contacts avec le monde extérieur.

L'AU sert de passerelle entre le MHS X.400 et un service de communication extérieur. Deux types d'AU ont été définis dans X.400-88: l'AU de télétex et l'AU de télex. Ces AUs permettent par exemple à un utilisateur qui n'a pas souscrit à un service de messagerie interpersonnelle d'émettre et de recevoir occasionnellement des IPMs sur son terminal de télétex ou télex. Mais cette possibilité ne se limite pas au message interpersonnel et peut s'appliquer à tout autre type de message.

Le PDAU est un cas particulier d'AU. C'est un AU qui est connecté à un service de livraison physique (*Physical Delivery Service* ou PDS) tel que le service postal. Le PDAU permet à un utilisateur du MHS X.400 d'émettre des messages qui seront livrés sous une forme physique (ex.: une copie du message sur papier) à des récepteurs qui se trouvent en-dehors du MHS X.400. Dans certains cas, le PDS peut renvoyer une notification de réception/non-réception à l'émetteur. Contrairement à l'AU, le fonctionnement du PDAU est à "sens unique": il peut recevoir des messages du MTS mais ne peut pas soumettre des messages au MTS.

2.6.3 Services de répertoire (*Directory services*) X.500

Nous avons vu dans 2.5.1 que l' *O/R Address* est un ensemble d'attributs qui permettent aux MTAs de trouver un chemin pour atteindre un utilisateur du MTS. Malheureusement, ces attributs sont nombreux et difficiles à mémoriser par un être humain. Un nom tel que

"Professeur Van Bastelaer des FUNDP"

est certainement un identifiant plus facile à retenir que l'exemple de la figure 2-7, mais il ne contient pas les informations nécessaires aux MTAs pour le routage.

Pour cette raison, CCITT introduit la notion de *Directory Name* dans X.400-88. Un *O/R Name* dans X.400-88 est soit une *O/R Address*, soit un *Directory Name*. Le *Directory Name* est un nom " *user-friendly*" qui identifie un utilisateur unique dans le MTS; par rapport à l' *O/R Address*, il possède les avantages suivants:

-Le *Directory Name* est plus facile à retenir par un être humain. C'est en général une chaîne de caractères qui spécifie l' **identité** de l'utilisateur et non la **localisation** de l'utilisateur comme c'est le cas de l' *O/R Address*.

-Le *Directory Name* est plus stable que l' *O/R Address*. Cette dernière est susceptible d'être modifiée au moindre changement de localisation de l'utilisateur.

Malgré ces avantages, le *Directory Name* a l'inconvénient de ne pas contenir les informations nécessaires au routage du message. Ce problème est résolu grâce aux services de répertoire X.500. X.500 est une Recommandation CCITT qui fournit des services qui permettent de traduire un *Directory Name* en une *O/R Address*. C'est une sorte de "bottin" électronique qui est, de l'avis de beaucoup de spécialistes, une des applications les plus utiles et les plus réussies du modèle OSI. Il est entièrement intégré dans X.400-88.

X.500 est basé sur deux composants: le *Directory User Agent* (DUA) et le *Directory Service Agent* (DSA). Le DUA est une interface qui permet à un utilisateur d'adresser des requêtes au système de répertoire (*Directory System*), et le DSA est responsable de stocker et de gérer un fragment d'informations de répertoire. Les DSAs forment le *Directory System* (Système de Répertoire), et les DUAs forment avec le *Directory System* le *Directory Environment* (Environnement de Répertoire). L'ensemble des informations de répertoire s'appelle la *Directory Information Base* (DIB), c'est une (très) grande base de données distribuée qui est gérée par les différents DSAs du système de répertoire. (Voir figure 2-9)

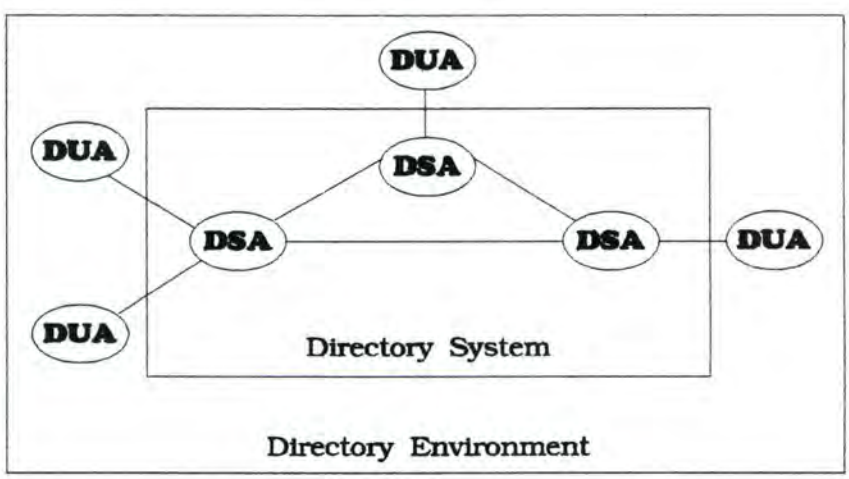

Figure 2-9 : Modèle fonctionnel des services de répertoire X.500

Lorsqu'un UA veut soumettre un message, il donne le *Directory Name* du récepteur au DUA, qui envoie une requête au DSA qui lui est connecté. Le DSA coopère avec les autres DSAs du *Directory System* pour trouver l' *O/R Address* correspondant et la renvoie au DUA. Celui-ci peut alors donner l' *O/R Address* à l'UA émetteur qui l'utilise pour soumettre le message au MTS.

Les MTAs du MTS peuvent également utiliser les services de répertoire X.500. Un UA qui n'a pas directement accès à un DUA peut se contenter de donner le *Directory Name* du récepteur comme *O/R Name* au MTS. Le MTS se chargera alors d'interroger le *Directory System* pour trouver l' *O/R Address* correspondant. L'UA peut également donner le *Directory Name* et l' *O/R Address* comme *O/R Name* au MTS. Dans ce cas-ci, le MTS va d'abord tenter d'utiliser l' *O/R Address* pour délivrer le message au récepteur. Si l' *O/R Address* s'avère incorrecte et l'opération échoue, alors le MTS interrogera le *Directory System* avec le *Directory Name* pour trouver l' *O/R Address* correspondant.

La figure 2-10 montre l'interconnexion entre le MHS X.400 et l'environnement de

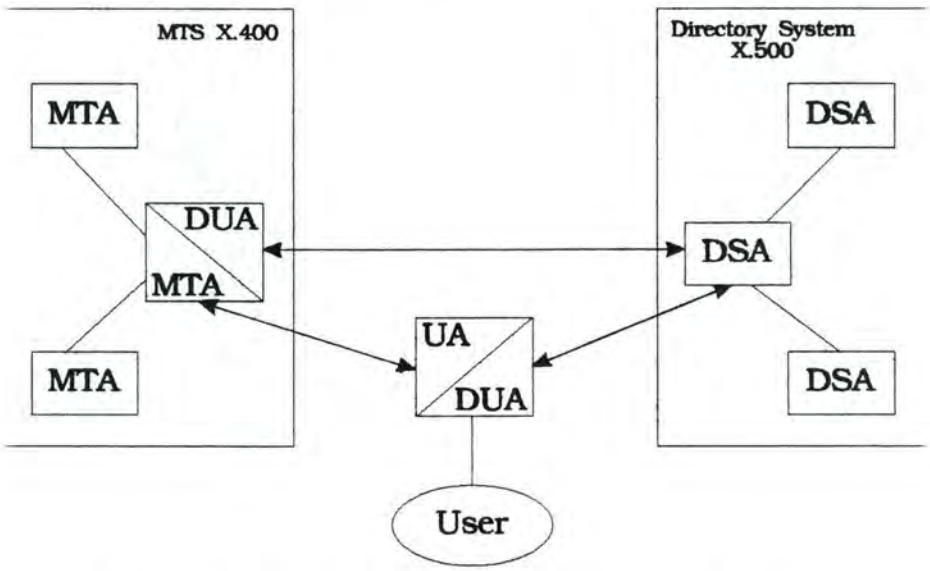

Figure 2-10 : Interconnexion entre le MHS X.400 et l'Environnement de Répertoire X.500

répertoire X.500.

2.6.4 Services de sécurité

Les réseaux de télécommunication sont vulnérables à des actions mal intentionnées et illégales. Dans X.400-84, aucune protection n'est fournie aux utilisateurs du MHS X.400 et ceux-ci n'ont aucune garantie concernant par exemple la bonne origine et l'intégrité du message reçu.

Pour cette raison, CCITT introduit 14 services de sécurité dans X.400-88. Ces services protègent le MHS X.400 contre les nombreuses menaces qui peuvent

survenir, comme par exemple l'interception et la modification d'un message par un intrus. Ces services de sécurité sont eux-mêmes basés sur 20 éléments de services qui sont décrits dans 10.2 de la Recommandation X.402. Les lecteurs intéressés peuvent trouver à l'Annexe E de la Recommandation X.402 la table de correspondance entre les services de sécurité et les éléments de service, montrant quels derniers sont utilisés par quels premiers.

Nous allons décrire brièvement les services de sécurité de X.400 tout en gardant leur nom en anglais pour permettre aux lecteurs de les retrouver plus facilement dans la série de Recommandations X.400: [X.400-88]

Message origin authentication: Permettre au récepteur, ou à un MTA à travers lequel passe le message, d'authentifier l'identité de l'émetteur du message.

Report origin authentication: Permettre à l'émetteur d'authentifier l'origine d'un rapport de livraison/non-livraison.

Probe origin authentication: Permettre à un MTA à travers lequel le *probe* passe d'authentifier l'origine du *probe.*

Proof of delivery: Permettre à l'émetteur d'un message d'authentifier le message délivré et son contenu, et l'identité du(des) récepteur(s).

Proof of submission: Permettre à l'émetteur d'un message d'authentifier que le message a été soumis au MTS pour être délivré au(x) récepteur(s) originellement spécifié(s).

Secure access management: Fournir l'authentification entre deux composants adjacents, et la mise en oeuvre du contexte de sécurité.

Content integrity: Permettre au récepteur de vérifier que le contenu original d'un message n'a pas été modifié.

Content confidentiality: Empêcher la divulgation non-autorisée du contenu d'un message par une tierce personne autre que le récepteur.

Message flow confidentiality: Permettre à l'émetteur d'un message de dissimuler le flux de messages à travers le MHS.

Message sequence integrity: Permettre à l'émetteur de fournir au récepteur la preuve que la séquence des messages a été préservée.

Non-repudiation of origin: Fournir au(x) récepteur(s) d'un message la preuve de l'origine du message et de son contenu, qui le(s) protègera contre toute tentative par l'émetteur de renier à tort l'envoi du message ou son contenu.

Non-repudiation of delivery: Fournir à l'émetteur d'un message la preuve de la livraison du message, qui le protègera contre toute tentative par le(s) récepteur(s) de renier à tort la réception du message ou son contenu.

Non-repudiation of submission: Fournir à l'émetteur d'un message la preuve de la soumission du message, qui le protègera contre toute tentative par le MTS de renier à tort le fait que le message a été soumis pour être délivré au(x) récepteur(s) originellement spécifié(s).

Message security labelling: Fournir la possibilité de catégoriser un message, indiquant sa sensibilité, laquelle détermine la manipulation du message dans la ligne de la politique de sécurité en cours.

Il est très important de comprendre la différence entre *proof* (preuve) et *non-repudiation* (non-répudiation). *Proof* peut être comparé à un retour de reçu signé, qui fournit une confirmation du bon déroulement de l'opération à son demandeur. *Non-repudiation* est une confirmation encore plus forte, elle peut être comparée à une signature notariée et sert à authentifier la signature du reçu. Pour répondre à une demande de *non-repudiation*, on doit obtenir un "certificat non-répudiable" de la part d'une autorité reconnue et le renvoyer à son demandeur comme une confirmation officielle et irréfutable. X.400-88 ne spécifie pas comment un certificat non-répudiable peut être obtenu, mais on peut imaginer les services de répertoire X.500 remplir le rôle de l'autorité.

Services de sécurité	MTS-user émetteur	MTS	MTS-user récepteur
Message origin authentication	P	U	U
Report origin authentication	U	P	-
Probe origin authentication	P	U	-
Proof of delivery	U	-	P
Proof of submission	U	P	-
Secure access management	P	U	P
Content integrity	P	-	U
Content confidentiality	P	-	U
Message flow confidentiality	P	-	-
Message sequence integrity	P	-	U
Non-repudiation of origin	P	-	U
Non-repudiation of submission	U	P	-
Non-repudiation of delivery	U	-	P
Message security labelling	P	U	U

P = Fournisseur du service (Provider)
U = Utilisateur du service

Table 2-2 : Fourniture et utilisation des services de sécurité X.400 [X.400-88]

La table 2-2 montre la liste complète des services de sécurité X.400, avec des indications sur le fournisseur et l'utilisateur de chacun de ces services.

Chapitre 3

Introduction à la Recommandation X.435

La Recommandation X.435 décrit le service de messagerie EDI défini par CCITT. Elle représente l'effort pour standardiser la rencontre entre l'EDI et X.400, et spécifie la manière dont le dernier doit être utilisé pour le premier. Cette Recommandation est parue en juin 1990, mais il s'agissait d'une version provisoire (*"Draft Recommendation"*) et était susceptible de changements futurs. L'ISO a cependant indiqué son intention de reconnaître X.435 comme étant un standard international dès que la version définitive sera disponible.

X.435 est en réalité décrit dans deux documents distincts: les Recommandations F.435 et X.435. La Recommandation F.435 décrit le fonctionnement général et les services offerts, et la Recommandation X.435 décrit le format des messages et le protocole utilisé.

X.435 est basé sur X.400-88 et utilise les services de transfert de message de celui-ci pour échanger des interchanges EDI. X.435 n'impose pas l'utilisation du codage EDIFACT, mais on espère bien entendu que X.435 sera le point de jonction entre le standard de communication et le standard de présentation de l'EDI: X.400 et EDIFACT.

Nous allons décrire dans ce chapitre le système de messagerie EDI, le format et l'utilisation du message EDI (EDIM) et de la notification EDI (EDIN), le principe de relais et la responsabilité EDI. Pour mieux mettre en évidence le problème de l'utilisation de X.400-84 pour X.435, nous laissons volontairement au chapitre suivant les discussions sur l' *EDI-Message Store*, les services de sécurité supplémentaires de X.435, et l'utilisation des services de répertoire X.500 par X.435.

3.1 Architecture générale de X.435

La figure 3-1 montre le modèle fonctionnel du service de messagerie EDI X.435.

Avant de commenter ce schéma, nous allons d'abord définir le vocabulaire utilisé dans X.435:

EDIMG : *EDI Messaging* (Messagerie EDI)
EDIME : *EDI Messaging Environment* (Environnement de messagerie EDI)
EDIMS : *EDI Messaging System* (Système de messagerie EDI)

EDI-UA : *EDI User Agent*
EDI-MS : *EDI Message Store*

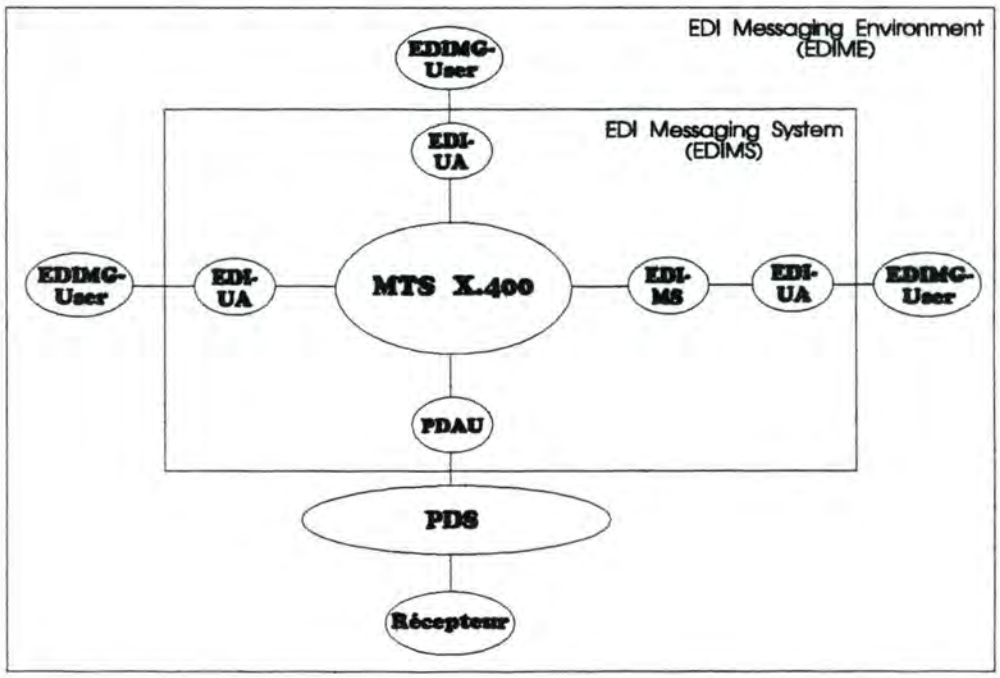

Figure 3-1 : Modèle fonctionnel du service de messagerie EDI X.435

Le *Message Transfer System* (MTS) est déjà décrit dans 2.2. Il s'agit ici de la version 1988 de X.400. Le MTS assure le service de transfert pour le message EDI (EDIM) et la notification EDI (EDIN).

L'EDIMS est formé des EDI-UAs, du MTS, et éventuellement des EDI-MS et des PDAUs. L'EDI-UA est un UA qui possède les fonctions spécifiques pour l'EDI, c'est une interface qui interagit directement avec l'EDIMG-user pour fournir à celui-ci les fonctions qui lui permettent d'envoyer et de recevoir des messages EDI. L'EDI-MS est un *Message Store* qui possède les fonctions spécifiques pour l'EDI, il est destiné à stocker les messages EDI reçus par le MTA et destinés à l'EDI-UA (Voir 4.1 pour plus de détails). Le PDAU est un PDAU normal comme nous l'avons déjà décrit dans 2.6.2; d'autres types d'AU (ex.: AU de fac-similé) pourraient faire l'objet d'une standardisation future. L'EDIMS est un système comparable au système de messagerie interpersonnelle (IPMS), dans ce sens que les deux systèmes utilisent les services du MTS X.400 pour transférer des messages. Mais si l'IPMS traite les messages (et notifications) interpersonnels, l'EDIMS traite quant à lui les messages (et notifications) EDI. Le protocole utilisé entre les EDI-UAs est appelé le protocole Pedi.

L'EDIME est formé des utilisateurs de messagerie EDI (EDIMG-user) et de l'EDIMS. L'EDIMG-user est en général un programme d'application EDI qui génère des interchanges EDI (des interchanges EDIFACT, par exemple) et qui utilise les services fournis par l'EDIMS pour les échanger avec d'autres EDIMG-users. Le

protocole utilisé entre les EDIMG-users est bien sûr leur protocole privé et ne fait pas partie de X.435.

Deux types de messages sont échangés dans l'EDIMS: le message EDI (EDIM) et la notification EDI (EDIN). Ces deux types de message sont appelés messages Pedi. Ils sont mis dans une enveloppe P1 et passent à travers le MTS X.400 pour leur transfert au récepteur (voir figure 3-2).

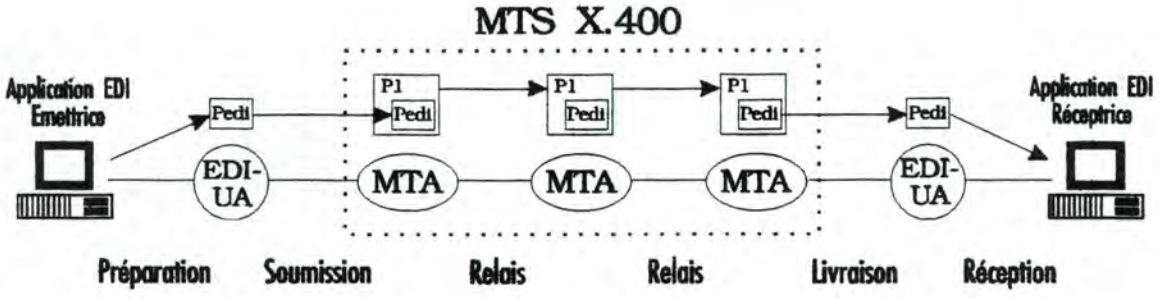

Figure 3-2 : Transfert de message Pedi par le service de messagerie X.400

Nous allons donner une description des différents champs qui composent l'EDIM et l'EDIN. La description sera assez détaillée, car une bonne compréhension de la structure de l'EDIM et l'EDIN sera nécessaire pour la suite de ce texte. Les lecteurs peuvent trouver à l'Annexe B la définition de l'EDIM et de l'EDIN en ASN.1.

3.2 Message EDI

L'EDIM est un message X.435 contenant un interchange EDI qu'un EDI-UA envoie aux autres EDI-UAs. Il est divisé en deux partie comme le montre la figure 3-3: l'en-tête (*heading*) et le corps (*body*). Le *heading* contient les informations qui permettent à l'EDI-UA récepteur de traiter l'EDIM. Le *body* contient un *body-part* primaire et éventuellement des *body-parts* additionnels. Un seul interchange EDI peut se trouver dans un EDIM: on doit construire plusieurs EDIMs si on a plusieurs interchanges EDI à envoyer.

3.2.1 *Heading* du message EDI

Le *heading* de l'EDIM contient les champs suivants:

This-EDIM

L'identifiant de l'EDIM qui fournit une identification globale et unique pour toujours.

Originator

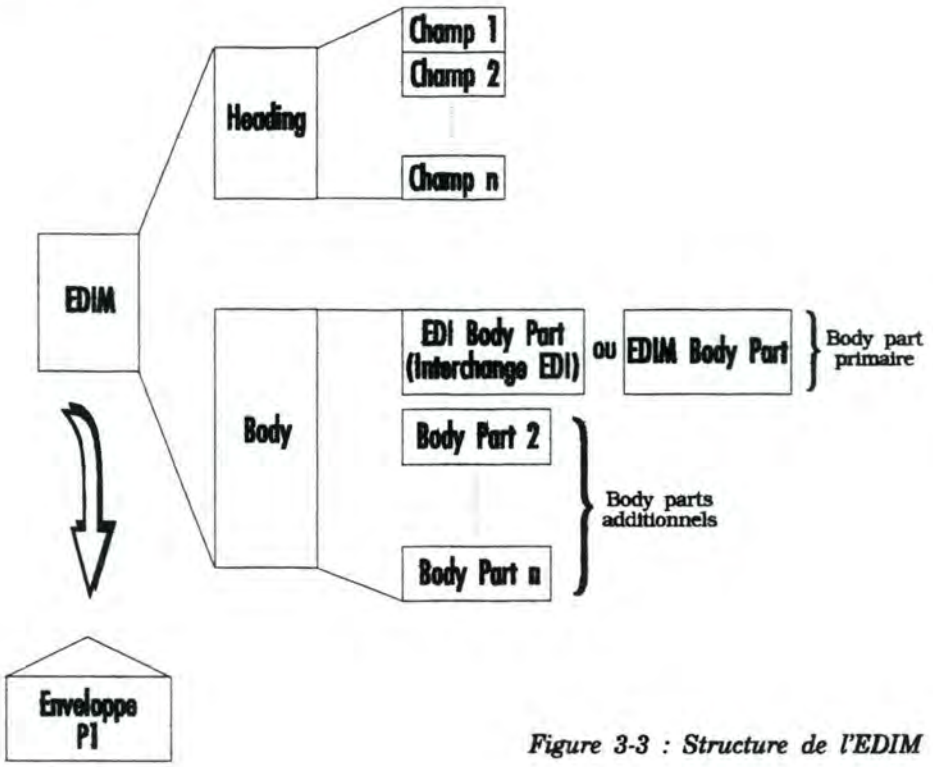

Figure 3-3 : Structure de l'EDIM

L'*O/R Name* de l'EDI-UA émetteur de l'EDIM.

Recipients

Un EDIM peut être envoyé à plus d'un récepteur. Cela peut être très utile lorsque l'interchange à envoyer est un appel d'offre, par exemple. Le champ ''Recipients'' contient donc les informations qui concernent chacun des EDI-UAs récepteurs. Les plus importantes de ces informations sont:

-Recipient: L'*O/R Name* du récepteur.

-EDI-notification-requests-field: Indications sur les EDINs que l'EDI-UA émetteur aimerait recevoir (''EDI-notification-requests''), le niveau de sécurité demandé sur les EDINs (''EDI-notification-security''), et le niveau de sécurité demandé sur le retour de l'EDIM (''EDI-reception-security).

-Responsibility-passing-allowed: Une valeur booléenne qui indique à l'EDI-UA récepteur la permission du passage de responsabilité. (Voir 3.4 pour la notion de responsabilité)

-Quelques champs copiés du segment UNB (voir 1.2.6) de l'interchange contenu dans l'EDIM s'il s'agit d'un interchange EDIFACT.

EDIN-receiver

L'identifiant de l'EDI-UA à qui les EDINs doivent être envoyées. Ce champ est utilisé lorsque l'EDI-UA émetteur demande des EDINs et que celles-ci sont à

renvoyer à un autre EDI-UA que celui de l'émetteur. Si aucune EDIN n'est demandée, alors ce champ ne doit pas être utilisé.

Le champ ''EDIN-receiver'' contient les informations suivantes:

-EDIN-receiver-name: L'*O/R Name* de l'EDI-UA récepteur des EDINs.

-Original-EDIM-identifier: L'identifiant de l'EDIM original. Il faut savoir qu'à chaque relais d'un EDIM, un nouvel EDIM portant un nouvel identifiant est créé pour contenir l'EDIM à relayer. L'identifiant de l'EDIM original est alors sauvegardé dans ce champ.

-First-recipient: L'*O/R Name* de l'EDI-UA à qui l'EDIM original est envoyé pour la première fois par l'UA émetteur. Autrement dit, l' *O/R Name* de l'EDI-UA qui aurait effectué le premier relais.

Responsibility-forwarded

Une valeur booléenne qui, lorsqu'elle est à ''vrai'', indique à l'EDI-UA récepteur que l'EDIM a été relayé sans acceptation de responsabilité.

EDI-bodypart-type

Indication sur le type de codage de l'interchange contenu dans l'EDIM. Les valeurs possibles sont EDIFACT, ANSI X.12, UNTDI, ... etc.

Incomplete-Copy

Une valeur booléenne qui, lorsqu'elle est à ''vrai'', indique à l'EDI-UA récepteur que l'EDIM a été relayé et qu'une partie de l'EDIM (des *body-parts*) a été retirée.

Expiry-Time

La date et l'heure d'expiration au-delà desquelles l'EDIM est considéré comme périmé.

Related-Messages

Les identifiants des EDIMs que l'émetteur considère être relatifs à l'EDIM envoyé.

Obsoleted-EDIMS

Les identifiants des EDIMs qui sont rendus périmés par l'EDIM envoyé.

EDI-application-security-elements

Champ qui permet d'avoir une sécurité de bout en bout (*end-to-end*) entre les applications EDI (EDIMG-users). (Voir 4.3.3 pour la sécurité de bout en bout dans X.435 et l'usage de ce champ)

Cross-referencing-information

Références aux différents *body-parts* additionnels contenus dans l'EDIM.

Les champs suivants du *heading* sont copiés des segments de service (voir 1.2.6) de l'interchange contenu dans l'EDIM s'il s'agit d'un interchange EDIFACT:

EDI-message-type

Indication sur le type de message EDI contenu dans l'EDIM. Pour un interchange EDIFACT, ce champ est copié du segment UNH. Les valeurs possibles sont INVOIC (facture), ORDERS (bon de commande), ... etc.

Service-string-advice

Indication sur les caractères de service utilisés dans l'interchange EDIFACT. Ce champ est copié du segment UNA.

Syntax-identifier

La syntaxe utilisée pour l'interchange EDIFACT. Ce champ est copié du segment UNB.

Interchange-sender

L'identifiant de l'émetteur de l'interchange. Ce champ est copié du segment UNB.

Date-and-time-of-preparation

La date et l'heure de préparation de l'interchange. Ce champ est copié du segment UNB.

Application reference

La référence générale de l'application qui a généré l'interchange. Ce champ est copié du segment UNB.

Quand nous disons que ces derniers champs sont ''copiés'' de l'interchange contenu dans l'EDIM, nous voulons dire que ces champs sont sémantiquement identiques aux éléments de donnée qui portent les mêmes noms et qui se trouvent dans les segments de service de l'interchange. Ces champs dédoublés avec l'interchange sont traduits en ASN.1 et mis dans le *heading* de l'EDIM pour permettre à l'EDI-UA récepteur de prendre des décisions sur l'interchange reçu sans devoir connaître la syntaxe EDIFACT de ce dernier. On remarque également que le champ ''Originator'' et certaines informations du champ ''Recipients'' sont également dédoublés avec l'enveloppe P1.

3.2.2 *Body* du message EDI

Le *body* de l'EDIM est divisé en deux parties: le *primary body part* et les *additional body parts*.

Primary body part

Le *primary body part* peut être de deux types: *EDI body part* ou *EDIM body part*. Dans le premier cas il s'agit d'un interchange EDI généré par une application EDI, et dans le second cas il s'agit d'un EDIM qui a été relayé.

Additional body parts

Les *additional body parts* sont des informations relatives au *primary body part*. Ces informations peuvent être de nature très différentes: dessins, télex, voix, ... etc. Les *additional-body-parts* peuvent optionnellement être référencés par le champ ''cross-referencing-information'' du *heading*. Cela facilite la recherche des *additional body parts* par l'EDI-UA comme le montre la figure 3-4.

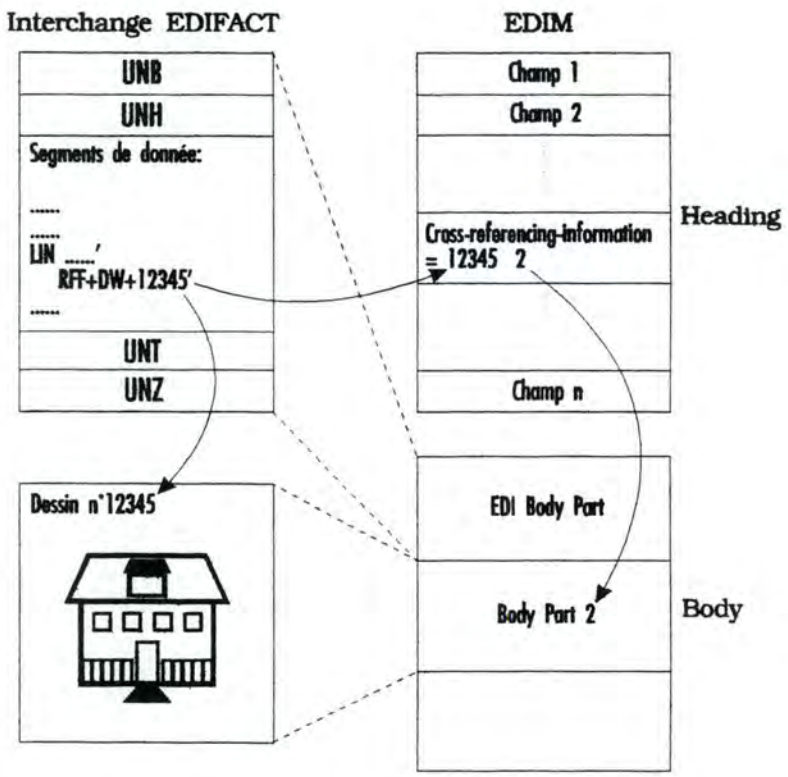

Figure 3-4 : "Cross-referencing" des body parts additionnels dans un EDIM

3.3 Notification EDI

L'EDIN est un message X.435 généré par l'EDI-UA (ou l'EDI-MS s'il existe) du récepteur d'un EDIM pour signifier à l'émetteur l'acceptation, le refus ou le relais de la responsabilité de l'EDIM (voir 3.4 pour la notion de responsabilité).

Il existe trois types d'EDIN:

-Notification positive (PN): La PN est une notification EDI que l'EDI-UA récepteur d'un EDIM envoie à l'EDI-UA émetteur pour signifier à ce dernier l'acceptation de la responsabilité de l'EDIM.

-Notification négative (NN): La NN est une notification EDI que l'EDI-UA récepteur d'un EDIM envoie à l'EDI-UA émetteur pour signifier à ce dernier le refus de la responsabilité de l'EDIM.

-Notification de relais (FN ou *forwarded notification*): La FN est une notification EDI que l'EDI-UA récepteur d'un EDIM envoie à l'EDI-UA émetteur pour signifier à ce dernier le relais de la responsabilité de l'EDIM à un autre EDI-UA.

Le récepteur de l'EDIN est l'EDI-UA émetteur de l'EDIM ou, s'il est présent, l'EDI-UA dont l'*O/R Name* se trouve dans le champ "EDIN-receiver" du *heading* de l'EDIM reçu. Chaque EDI-UA récepteur d'un EDIM ne peut renvoyer qu'une seule EDIN (PN, NN ou FN), et le dernier EDI-UA récepteur ne peut renvoyer qu'une PN ou NN. Cela signifie que la séquence des EDINs reçues par l'EDI-UA émetteur a la forme suivante:

FN, FN, ... , FN, PN

ou

FN, FN, ... , FN, NN

Les structures des trois types d'EDIN partagent une partie commune, appelée les champs communs (*common-fields*). L'autre partie est spécifique à chaque type d'EDIN. (Voir figure 3-5)

3.3.1 Champs communs de la notification EDI

Les *common-fields* de l'EDIN comporte les champs suivants:

Subject-EDIM

L'identifiant de l'EDIM auquel l'EDIN fait référence.

Ce champ est copié de l'EDIM reçu. Soit du champ "this-EDIM" du *heading*, soit du

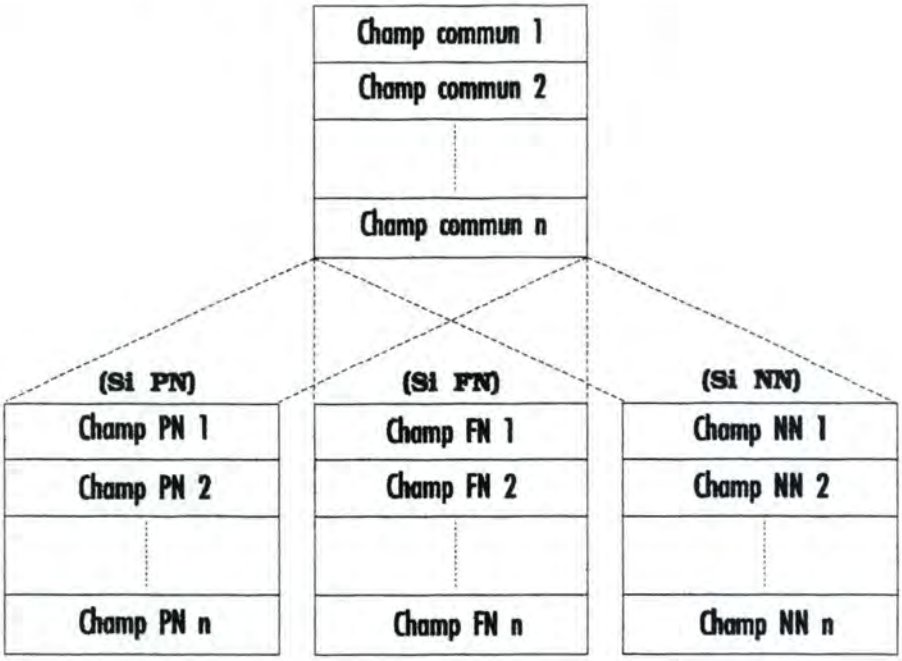

Figure 3-5 : Structure de l'EDIN

champ ''EDIN-receiver'' du *heading* si l'EDIM original a été relayé.

EDIN-originator

L'*O/R Name* de l'EDI-UA générateur de l'EDIN. Autrement dit, l'EDI-UA récepteur de l'EDIM.

First-recipient

L'*O/R Name* du premier EDI-UA récepteur dans la chaîne de relais. Autrement dit, l'EDI-UA du récepteur original (*intended recipient)* de l'EDIM original.

Ce champ permet, avec le champ ''subject-EDIM'', à l'EDI-UA émetteur de faire la corrélation entre l'EDIN et l'EDIM original.

Notification-time

La date et l'heure auxquelles l'EDIN est générée.

Notification-security-elements

Champ comportant des sous-champs qui permettent de retourner une copie de l'EDIM reçu (''original-content''), de retourner une copie de l'EDIM reçu sous forme encryptée (''original-content-integrity-check''), et d'avoir une sécurité de bout en bout (*end-to-end*) entre les EDIMG-users (''EDI-application-security-elements''). (Voir 4.3.3 pour la sécurité de bout en bout dans X.435 et l'usage de ce champ)

EDIN-initiator

Indication sur le niveau auquel l'EDIN est générée. Trois valeurs sont définies pour ce champ:

-internal-ua: L'EDI-UA génère l'EDIN soit pour des raisons locales, soit parce que la génération de l'EDIN lui a été déléguée par l'EDIMG-user.

-external-ua: L'EDI-UA génère l'EDIN à la demande de l'EDIMG-user.

-internal-ms: L'EDI-MS génère l'EDIN soit pour des raisons locales, soit parce que la génération de l'EDIN lui a été déléguée par l'EDIMG-user.

3.3.2 Champs spécifiques de la notification EDI

Certains champs de l'EDIN sont spécifiques au type d'EDIN.

1) Le seul champ spécifique à la PN est:

PN-supplementary-information

Information supplémentaire destinée à l'EDI-UA récepteur de l'EDIN pour clarifier la PN.

2) Les champs spécifiques à la NN sont:

NN-reason-code

Un code qui précise la raison de la génération de la NN. Les codes sont classés en trois catégories: code de l'EDI-UA ou l'EDI-MS (NN-ua-ms-reason-code), code de l'utilisateur (NN-user-reason-code), ou code du PDAU (NN-pdau-reason-code). Chaque code est en réalité composé d'un code de base (*basic code*) et d'un code de diagnostic (*diagnostic code*).

NN-supplementary-information

Information supplémentaire destinée à l'EDI-UA récepteur de l'EDIN pour clarifier la NN.

3) Les champs spécifiques à la FN sont:

Forwarded-to

L'*O/R Name* du nouvel EDI-UA récepteur à qui l'EDIM reçu est relayé.

FN-reason-code

Un code qui précise la raison de la génération de la FN. Les codes sont classés en trois catégories: code de l'EDI-UA ou l'EDI-MS (FN-ua-ms-reason-code), code de l'utilisateur (FN-user-reason-code), ou code du PDAU (FN-pdau-reason-code). Chaque code est en réalité composé d'un code de base (*basic code*) et d'un code de diagnostic (*diagnostic code*).

FN-supplementary-information

Information supplémentaire destinée à l'EDI-UA récepteur de l'EDIN pour clarifier la FN.

3.4 Responsabilité et relais d'EDIM

3.4.1 Principes de base

Le relais est un concept très utile dans un service de messagerie EDI. En effet, il arrive souvent dans une société qu'un EDI-UA sert de centralisateur des messages EDI reçus pour ensuite les distribuer aux EDI-UAs finals. Par exemple, une société peut avoir un EDI-UA central qui relaie les EDIMs reçus aux départements de vente, d'achat, de comptabilité, de développement, ... etc selon le type d'interchange qui se trouve dans l'EDIM. Cela permet à la société d'avoir une adresse EDI unique et facilite ainsi la communication avec ses partenaires EDI.

Dans X.435, le relais est généralement effectué par l'EDI-UA, mais aussi par l'EDI-MS s'il existe. On peut relayer un EDIM à plusieurs autres récepteurs. Mais pour la suite de ce texte, nous allons supposer que c'est l'EDI-UA qui effectue le relais, et à un seul EDI-UA récepteur seulement.

Lorsqu'un EDI-UA décide de relayer un EDIM à un autre EDI-UA, il construit un nouvel EDIM en mettant la totalité du message reçu - c'est-à-dire l'EDIM reçu et optionnellement son enveloppe P1 - dans le *primary body part* du *body* du nouvel EDIM. Ce *primary body part* est dit de type *EDIM body part*. De ce fait, on obtient une sorte d'EDIM "récursif" qui peut comporter plusieurs niveaux de récursivité si l'EDIM original a été relayé plusieurs fois. Par convention, l'EDIM reçu qui est à relayer est appelé **EDIM-sujet** (*subject EDIM*), le nouvel EDIM construit pour contenir l'EDIM-sujet est appelé **EDIM-relayé** (*forwarded EDIM*), alors que l'EDIM initialement émis par l'EDI-UA émetteur est appelé **EDIM original** (voir figure 3-6). Un EDIM-relayé devient un nouvel EDIM-sujet pour le nouveau récepteur si celui-ci décide de le relayer encore une fois.

L'EDIMS inclut un autre concept important: la responsabilité EDI. C'est un concept qui est lié à la génération d'EDIN et qui concerne la manière dont l'EDI-UA traite un EDIM. La responsabilité peut être acceptée, refusée ou relayée par l'EDI-UA récepteur de l'EDIM.

Lorsqu'un EDIM est **rendu disponible** ("*made available*" en terme original) à l'EDIMG-user par l'EDI-UA récepteur, alors ce dernier est obligé d'accepter la

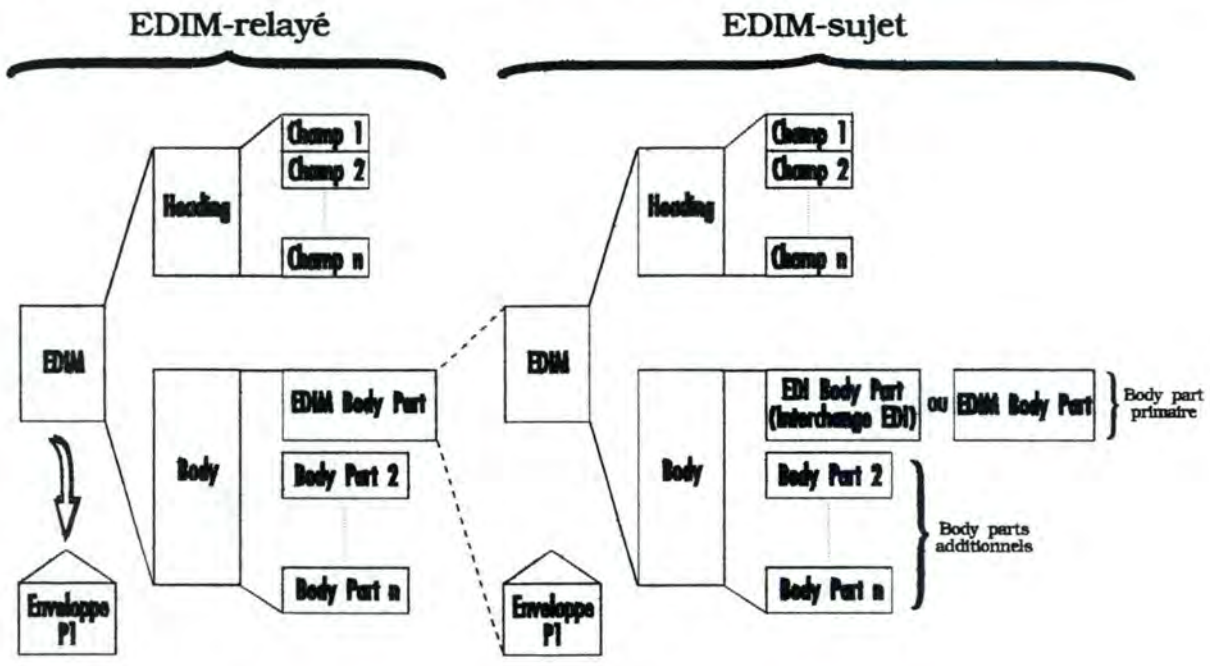

Figure 3-6 : Structure de l'EDIM-relayé (forwarded EDIM)

responsabilité. Par le terme ''rendu disponible'' nous entendons que:

1) L'EDIMG-user a pu consulter le contenu du *body* de l'EDIM reçu, ou

2) L'EDI-UA a enlevé/ajouté des *body parts* de/à l'EDIM reçu.

Acceptation de la responsabilité

L'acceptation de la responsabilité implique que l'EDI-UA récepteur envoie, si cela a été demandé, une PN à l'EDI-UA émetteur de l'EDIM original pour signifier son acceptation de l'EDIM. S'il le désire, l'EDI-UA récepteur peut ensuite relayer l'EDIM à un autre EDI-UA. Mais dans ce cas-ci il ne peut pas générer une FN, car l'EDIM est considéré comme avoir atteint le récepteur final suite à l'acceptation de la responsabilité.

Dans le cas du relais après l'acceptation de responsabilité, l'EDIM-relayé (contenant l'EDIM-sujet) devient un nouvel EDIM original et l'EDI-UA qui le relaie devient l'émetteur. Une nouvelle relation de responsabilité est alors établie et toutes les EDINs qui seront générées par la suite deviendront une affaire entre les futurs EDI-UAs récepteurs et le nouvel EDI-UA émetteur. Avant le relais, l'EDI-UA peut enlever/ajouter des *body parts* de/à l'EDIM. Il y a cependant des règles concernant le relais de l'EDIM après acceptation de responsabilité:

1) L'EDI-UA ne peut enlever un *primary body part* s'il est de type *EDIM body part.*

2) L'EDI-UA ne peut **modifier** les *body parts.*

3) Si l'EDI-UA enlève un *primary body part* de type *EDI body part*, il doit le remplacer par un *body part* bidon nommé *removed-edi-body* pour marquer son absence.

4) Si l'EDI-UA enlève un *additional body part*, il doit le remplacer par un *place holder* pour indiquer le type d' *additional body part* enlevé.

Refus de la responsabilité

Le refus de la responsabilité implique que l'EDI-UA récepteur envoie, si cela a été demandé, une NN à l'EDI-UA émetteur de l'EDIM original pour signifier son refus de l'EDIM.

Relais de la responsabilité

Le relais de la responsabilité implique que l'EDI-UA récepteur envoie, si cela a été demandé, une FN à l'EDI-UA émetteur de l'EDIM original pour signifier le relais de l'EDIM. L'EDIM ne peut être rendu disponible à l'EDIMG-user et doit rester inchangé avant son relais.

Nous allons voir les différents cas de figure possibles du transfert de l'EDIM. Dans les exemples qui vont suivre, nous supposons que toutes les trois EDINs (PN, NN et FN) sont requises lors de l'envoi de l'EDIM.

3.4.2 Acceptation ou refus de la responsabilité sans relais

Dans ce cas de base, aucun relais de l'EDIM n'est effectué. La figure 3-7 illustre un

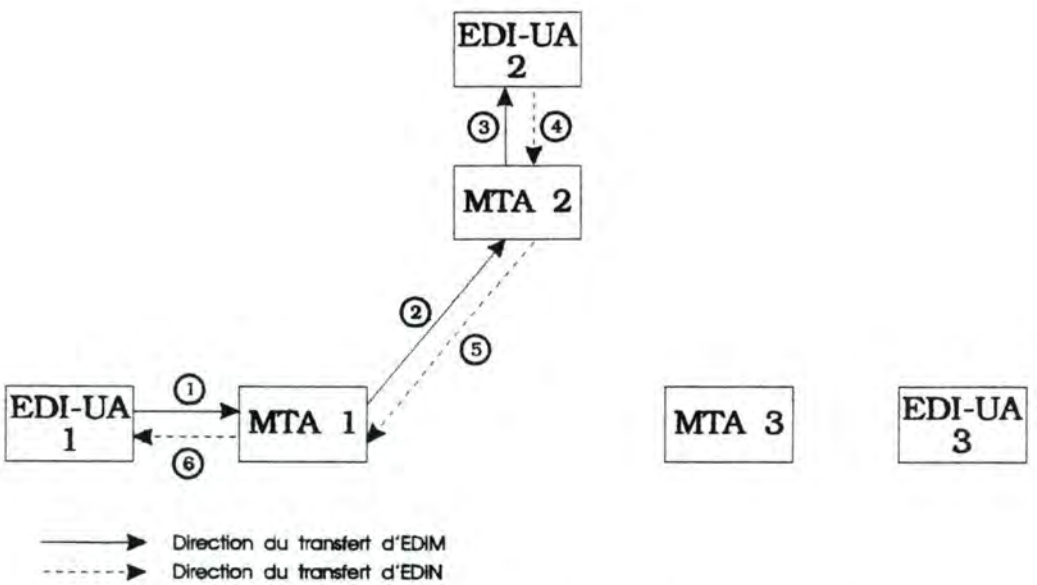

Figure 3-7 : Acceptation ou refus de la responsabilité

exemple, dans lequel l'EDIM préparé par l'EDI-UA 1 est adressé à l'EDI-UA 2.

Voici les différentes étapes de l'exemple:

Envoi de l'EDIM

1) L'EDI-UA 1 soumet l'EDIM au MTA 1.

2) Le MTA 1 transfère l'EDIM au MTA 2.

3) Le MTA 2 délivre l'EDIM à l'EDI-UA 2.

Envoi de l'EDIN

4) EDI-UA 2 soumet une EDIN (PN s'il accepte la responsabilité, NN s'il la refuse) au MTA 2.

5) Le MTA 2 transfère l'EDIN au MTA 1.

6) Le MTA 1 délivre l'EDIN à l'EDI-UA 1.

3.4.3 Relais de la responsabilité

Cette fois-ci, l'EDI-UA récepteur n'accepte pas la responsabilité et relaie celle-ci à un autre EDI-UA. La figure 3-8 illustre un exemple, dans lequel l'EDIM préparé par l'EDI-UA 1 est initialement adressé à l'EDI-UA 2, mais celui-ci le relaie à l'EDI-UA 3 sans accepter la responsabilité. Voici les différentes étapes de l'exemple (les étapes ne sont pas nécessairement exécutées dans l'ordre montré):

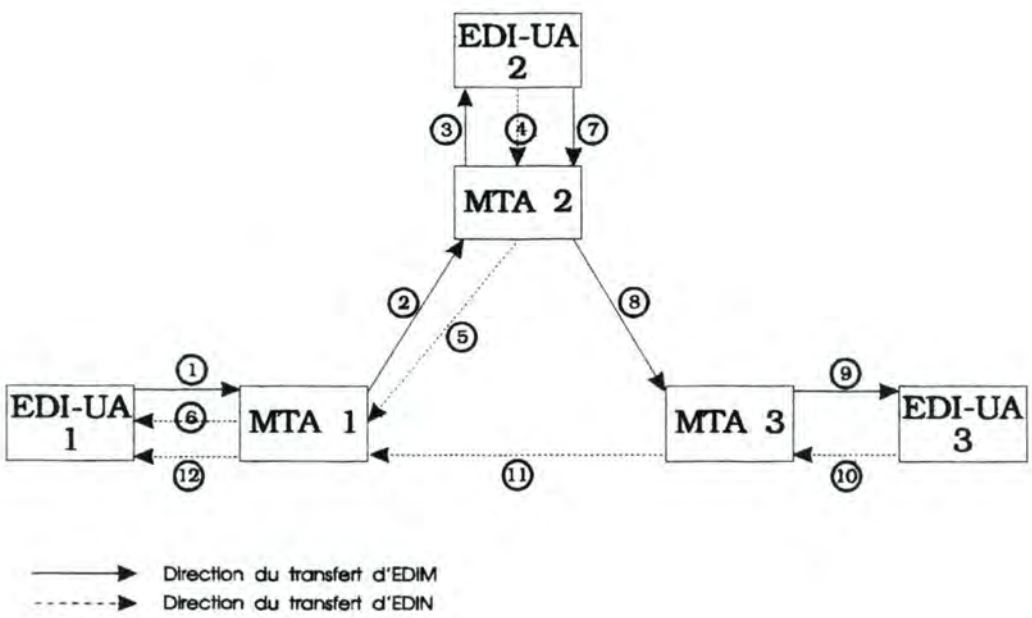

Figure 3-8 : Relais de la responsabilité

Envoi de l'EDIM

1) à 3): Voir 3.4.1.

Envoi de la FN

4) EDI-UA 2 soumet une FN au MTA 2.

5) Le MTA 2 transfère la FN au MTA 1.

6) Le MTA 1 délivre la FN à l'EDI-UA 1.

Relais de l'EDIM

7) L'EDI-UA 2 laisse l'EDIM-sujet inchangé, construit un EDIM-relayé et soumet celui-ci au MTA 2.

8) Le MTA 2 transmet l'EDIM au MTA 3.

9) Le MTA 3 délivre l'EDIM à l'EDI-UA 3.

Envoi de l'EDIN

10) EDI-UA 3 soumet une EDIN (PN s'il accepte la responsabilité, NN s'il la refuse, ou FN s'il la relaie à son tour) au MTA 3.

11) Le MTA 3 transfère l'EDIN au MTA 1.

12) Le MTA 1 délivre l'EDIN à l'EDI-UA 1.

3.4.4 Acceptation de la responsabilité avec relais de l'EDIM

Cette fois-ci, l'EDI-UA récepteur accepte la responsabilité mais relaie l'EDIM à un autre EDI-UA. La figure 3-9 illustre un exemple, dans lequel l'EDIM préparé par l'EDI-UA 1 est adressé à l'EDI-UA 2. Celui-ci, après avoir accepté la responsabilité, le relaie à l'EDI-UA 3. Voici les différentes étapes de l'exemple (les étapes ne sont pas nécessairement exécutées dans l'ordre montré):

Envoi de l'EDIM

1) à 3): Voir 3.4.1.

Envoi de la PN

4) EDI-UA 2 soumet la PN au MTA 2.

5) Le MTA 2 transfère la PN au MTA 1.

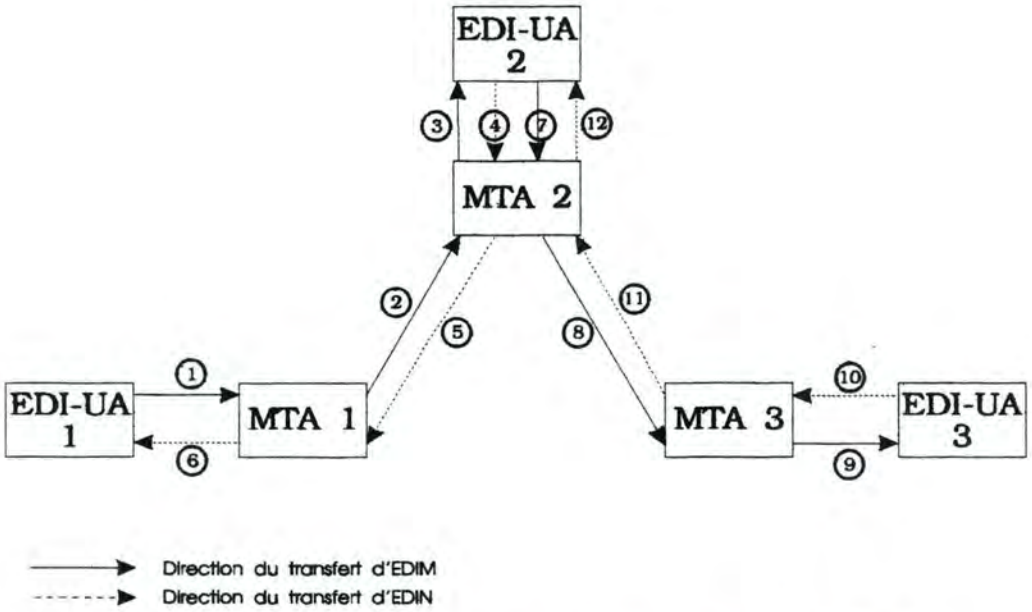

Figure 3-9 : Acceptation de la responsabilité avec relais de l'EDIM

6) Le MTA 1 délivre la PN à l'EDI-UA 1.

Relais de l'EDIM

7) L'EDI-UA 2 peut enlever/ajouter des *body parts* de/à l'EDIM-sujet, construit un EDIM-relayé et soumet celui-ci au MTA 2.

8) Le MTA 2 transmet l'EDIM au MTA 3.

9) Le MTA 3 délivre l'EDIM à l'EDI-UA 3.

Envoi de l'EDIN

10) EDI-UA 3 soumet une EDIN (PN s'il accepte la responsabilité, NN s'il la refuse, ou FN s'il la relaie à son tour) au MTA 3.

11) Le MTA 3 transfère l'EDIN au MTA 2.

12) Le MTA 1 délivre l'EDIN à l'EDI-UA 2.

3.4.5 Construction de l'EDIM-relayé

Nous savons que, lorsqu'un EDI-UA décide de relayer un EDIM à un autre EDI-UA, il construit un nouvel EDIM appelé EDIM-relayé en mettant la totalité du message reçu - c'est-à-dire l'EDIM-sujet et optionnellement son enveloppe P1 - dans le *primary body part* du *body* de l'EDIM-relayé. Nous savons également qu'un EDIM peut être relayé plusieurs fois avant qu'un EDI-UA final accepte la responsabilité.

Ces principes posent alors deux problèmes:

1) Il faut que les EDI-UAs qui relaient l'EDIM et l'EDI-UA final qui accepte l'EDIM sachent à qui envoyer les EDINs.

2) Il faut que l'EDI-UA émetteur puisse faire la corrélation entre les EDINs reçues et l'EDIM original.

Pour résoudre ces problèmes, X.435 spécifie des règles de construction de l'EDIM-relayé dans lesquelles le champ "EDIN-receiver" du *heading* de l'EDIM a une importance primordiale. Voici les plus importantes de ces règles qui concernent les champs du *heading* de l'EDIM-relayé:

A) This-EDIM: Un nouvel identifiant doit être généré.

B) Originator: L' *O/R Name* de l'EDI-UA qui relaie l'EDIM-sujet.

C) Recipients: L' *O/R Name* de l'EDI-UA à qui l'EDIM-sujet doit être relayé.

D) EDIN-receiver: Si ce champ est présent dans l'EDIM-sujet, il doit être copié tel quel à l'EDIM-relayé. S'il est absent ou si certains sous-champs de ce champ sont absents, alors on doit procéder de la manière suivante:

D1) Le sous-champ "EDIN-receiver-name": L' *O/R Name* de l'EDI-UA émetteur de l'EDIM-sujet.

D2) Le sous-champ "Original-edim-identifier": Le champ "This-EDIM" de l'EDIM-sujet.

D3) Le sous-champ "First-recipient": L' *O/R Name* de l'EDI-UA qui relaie l'EDIM-sujet.

La figure 3-10 illustre la manière dont ces règles sont appliquées. Sur la figure, l'EDI-UA 1 envoie un EDIM à l'EDI-UA 2, qui n'accepte pas la responsabilité et qui la relaie à l'EDI-UA 3. On voit bien que les règles de construction de l'EDIM-relayé permettent de sauvegarder l' *O/R Name* de l'EDI-UA 1, l'identifiant de l'EDIM original et l' *O/R Name* du premier récepteur dans le champ "EDIN-receiver". Ces informations sont ensuite utilisées par l'EDI-UA 3 pour envoyer la PN à l'EDI-UA 1. De plus, elles permettent à l'EDI-UA 1 de faire la corrélation entre l'EDIM original et les EDINs reçues, et de connaître ainsi le parcours de l'EDIM émis.

3.5 Services de messagerie EDI

L'EDIMS fournit des services de messagerie EDI (*EDI messaging services*) qui permettent à un EDIMG-user d'envoyer et de recevoir des EDIMs et des EDINs. Les services de messagerie EDI sont basés sur:

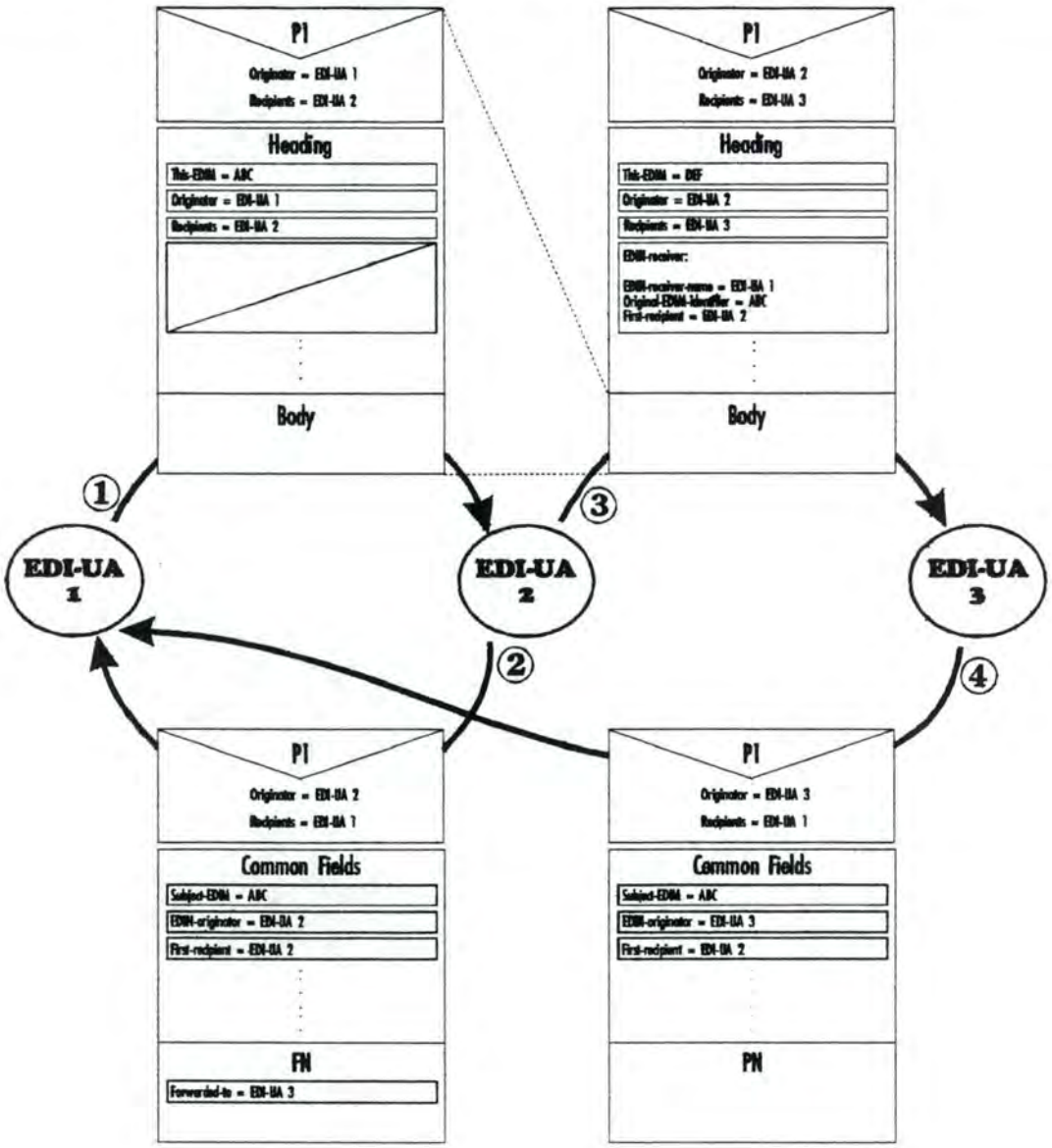

Figure 3-10 : Construction de l'EDIM-relayé et renvoi des EDINs

1) Les éléments de service MT version 1988.

2) Les éléments de service spécifiques à l'EDIMS.

L'ensemble de ces éléments de services forment les éléments de service de messagerie EDI (*EDI service elements*).

Les éléments de service MT version 1988 sont discutés à deux endroits dans ce texte: ceux qui sont communs aux deux versions de X.400 ont été décrits dans 2.2.3, et ceux qui sont spécifiques à X.400-88 seront décrits dans 4.5 lorsqu'on discutera de la difficulté de l'utilisation de X.400-84 pour X.435. Quant aux éléments de service spécifiques à l'EDIMS, ils sont au nombre de 30 et ce serait long et fastidieux de les décrire tous ici. Nous allons donc nous contenter de les citer, d'autant que beaucoup d'entre eux sont relatifs aux différents champs de donnée de l'EDIM et de l'EDIN et

que ceux-ci sont déjà expliqués dans 3.2 et 3.3. Les lecteurs peuvent néanmoins trouver à l'Annexe B de [F.435] la liste complète de ces éléments de service ainsi que leur fonction.

Les éléments de service spécifiques à l'EDIMS sont:

-Application Security Element
-Character Set
-Cross Reference Information
-EDI Forwarding
-EDI Message Type(s)
-EDI Notification Request
-EDI Standard Indication
-EDI-message Identification
-EDIM Responsibility Forwarding Allowed Indication
-EDIN receiver
-Expiry Date/Time Indication
-Incomplete Copy Indication
-Interchange Header
-Multi-part Body
-Non-repudiation of Content Originated (voir 4.3.2)
-Non-repudiation of Content Received (voir 4.3.2)
-Non-repudiation of Content Received Request
-Non-repudiation of EDI Notification (voir 4.3.2)
-Non-repudiation of EDI Notification Request
-Obsoleting Indication
-Originator Indication
-Proof of Content Received (voir 4.3.2)
-Proof of Content Received Request
-Proof of EDI Notification (voir 4.3.2)
-Proof of EDI Notification Request
-Recipient Indication
-Related Message(s)
-Services Indication
-Stored EDI Message Auto-forward
-Typed Body

Chapitre 4

Analyse de Faisabilité de l'Utilisation de X.400-84 pour X.435

L'utilisation de X.435 pour l'échange de document EDI est certainement la solution d'avenir. Trop de protocoles sont utilisés aujourd'hui et des compagnies et entreprises sont parfois forcées de faire appel aux coûteux services de VAN (*Value Added Network*) pour s'échanger des documents EDI. Puisqu'un protocole de communication pour EDI est enfin défini par CCITT, il y a fort à parier que tout le monde se tournera un jour vers cette solution.

Au moment où ce texte est rédigé, X.400-84 commence à peine à se généraliser. Quant à X.400-88, des outils de développement commencent seulement à être disponibles aux développeurs et des applications X.400-88 sont encore quasi inexistantes. Nous savons que X.435 utilise X.400-88 et non X.400-84, mais le retard de la généralisation de X.400-88 et le manque d'outils de développement de X.400-88 connus et fiables nous empêchent de développer immédiatement une application X.435. Nous sommes bien sûr tentés d'utiliser X.400-84 pour contourner ce problème, mais sachant que X.400-84 offre moins de fonctionnalités que X.400-88, cela n'est certainement pas sans risque.

Pour toutes ces raisons, nous avons pris la décision d'analyser si l'utilisation de X.400-84 pour X.435 est malgré tout possible. Autrement dit, analyser si l'échange EDI utilisant X.400-84 et basé sur X.435 est raisonnable et faisable. Nous essayons ainsi de trouver les problèmes que nous pourrions rencontrer et les précautions à prendre à cause de ce mariage de raison quelque peu forcé.

> Note: L'abréviation suivante sera utilisée tout au long de ce texte:
>
> X.435/X.400-84 = Echange EDI utilisant X.400-84 et basé sur X.435

Cette analyse est présentée dans ce chapitre en cinq parties. Nous allons discuter successivement de l'absence du *Message Store* dans X.400-84, de la difficulté éventuelle d'intégration de X.500 dans X.400-84, de l'absence des services de sécurité dans X.400-84, des actions spéciales à prendre quand on soumet un EDIM à un MTA version 84, et des éléments de service MT manquants de X.400-84 pour réaliser X.435.

4.1 Absence du *Message Store* dans X.400-84

Le *Message Store* (MS) est certainement l'amélioration la plus significative introduite dans X.400-88 (voir 2.6.1). Pour rappel, le MS est un composant optionnel associé à l'UA d'un utilisateur. Il permet de stocker les messages en entrée qui ne peuvent être délivrés immédiatement du MTA à l'UA à cause, par exemple, de la coupure de la machine sur laquelle se trouve l'UA. De plus, le MS permet à l'utilisateur d'effectuer des recherches sélectives des messages suivant les valeurs des attributs tels que le type du contenu (*content type*). Le MS est également capable d'effectuer certaines *auto-actions* - le relais automatique d'un message par exemple - sans l'intervention de l'utilisateur.

X.435 définit des possibilités additionnelles spécifiques à l'EDI qui peuvent optionnellement être ajoutées à un MS. Ces possibilités sont:

-La gestion des attributs MS spécifiques à l'EDI.

-L'*auto-action "EDI auto-forward"* (relais automatique de l'EDIM) avec ou sans acceptation de responsabilité.

Un MS ayant de telles capacités est qualifié d'EDI-MS. Nous allons voir quelles sont les fonctions d'un EDI-MS.

4.1.1 Attributs MS spécifiques à l'EDI

Nous avons vu dans 2.6.1 que X.400-88 définit un ensemble de 42 attributs généraux qu'un MS doit gérer pour chaque message P1 qu'il reçoit du MTA.

X.435 introduit 61 (!) attributs additionnels spécifiques à l'EDI (voir 18.7 et tables 1 & 2 dans [X.435]) qu'un EDI-MS doit gérer pour chaque message P1 en entrée qui contient un EDIM ou une EDIN. Voici quelques-uns des ces attributs additionnels spécifiques à l'EDI:

-originator
-edi-bodypart-type
-date-and-time-of-preparation
-expiry-time
.
.
.

Ainsi, un utilisateur peut par exemple demander, via son EDI-UA, à l'EDI-MS de lui fournir tous les EDIMs dont *originator* = la firme X, *edi-bodypart-type* = EDIFACT, *date-and-time-of-preparation* < = 15-08-1992, et *expiry-time* = date du jour. Grâce à ces attributs additionnels, un utilisateur peut effectuer des recherches sélectives qui sont propres à l'EDI.

La figure 4-1 montre comment une entrée d'attributs est constituée à partir d'un EDIM reçu. Chose assez surprenante, les attributs additionnels spécifiques à l'EDI sont si complets qu'on constate que la quasi totalité des champs de donnée de l'EDIM s'y retrouvent. Autrement dit, les différents champs des différents niveaux d'imbrication de l'EDIM sont décomposés, mis au même niveau et stockés ensuite dans l'*information-base.*

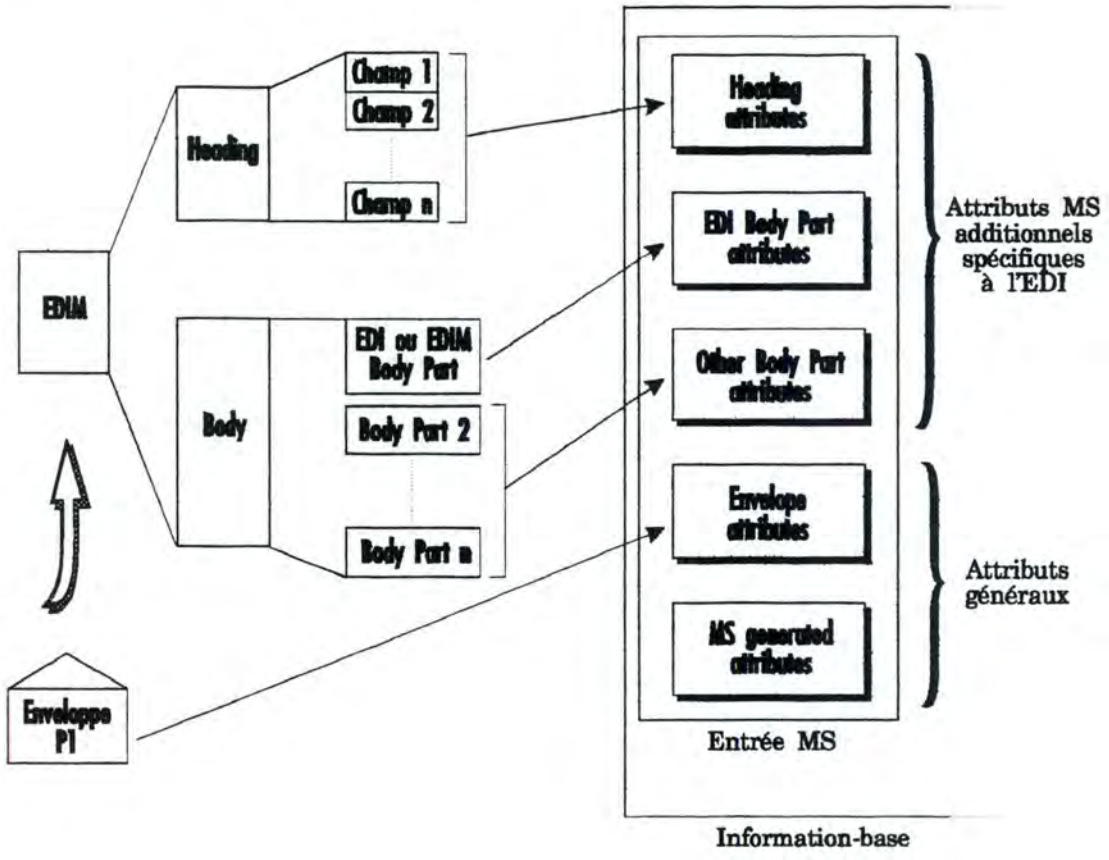

Figure 4-1 : Constitution d'une entrée MS à partir d'un EDIM

Dans le cas d'un EDIM-relayé (voir 3.4.1), les choses sont un peu différentes. Pour rappel, un EDIM-relayé est un EDIM qui en contient un autre, car l'EDIM reçu a été mis dans un EDIM-relayé pour être relayé à un autre récepteur. Le processus peut se répéter plusieurs fois et on obtient finalement une sorte d'EDIM ''récursif''. Pour un EDIM-relayé, l'EDI-MS doit garder une entrée principale (*main entry*) plus une ou plusieurs entrées-filles (*child entries*) liées entre elles par des pointeurs. Le nombre d'entrées-filles est égal au nombre de fois que l'EDIM original a été relayé, et la dernière entrée-fils correspond à l'EDIM original. La figure 4-2 montre comment les entrées d'attributs sont constituées dans l' *information-base pour* un EDIM-relayé.

4.1.2 *Auto-action ''EDI auto-forward''*

Le concept d'*auto-action* est déjà décrit dans 2.6.1. Rappelons que l' *auto-action* permet à l'UA de demander au MS de relayer automatiquement (*auto-forward*)

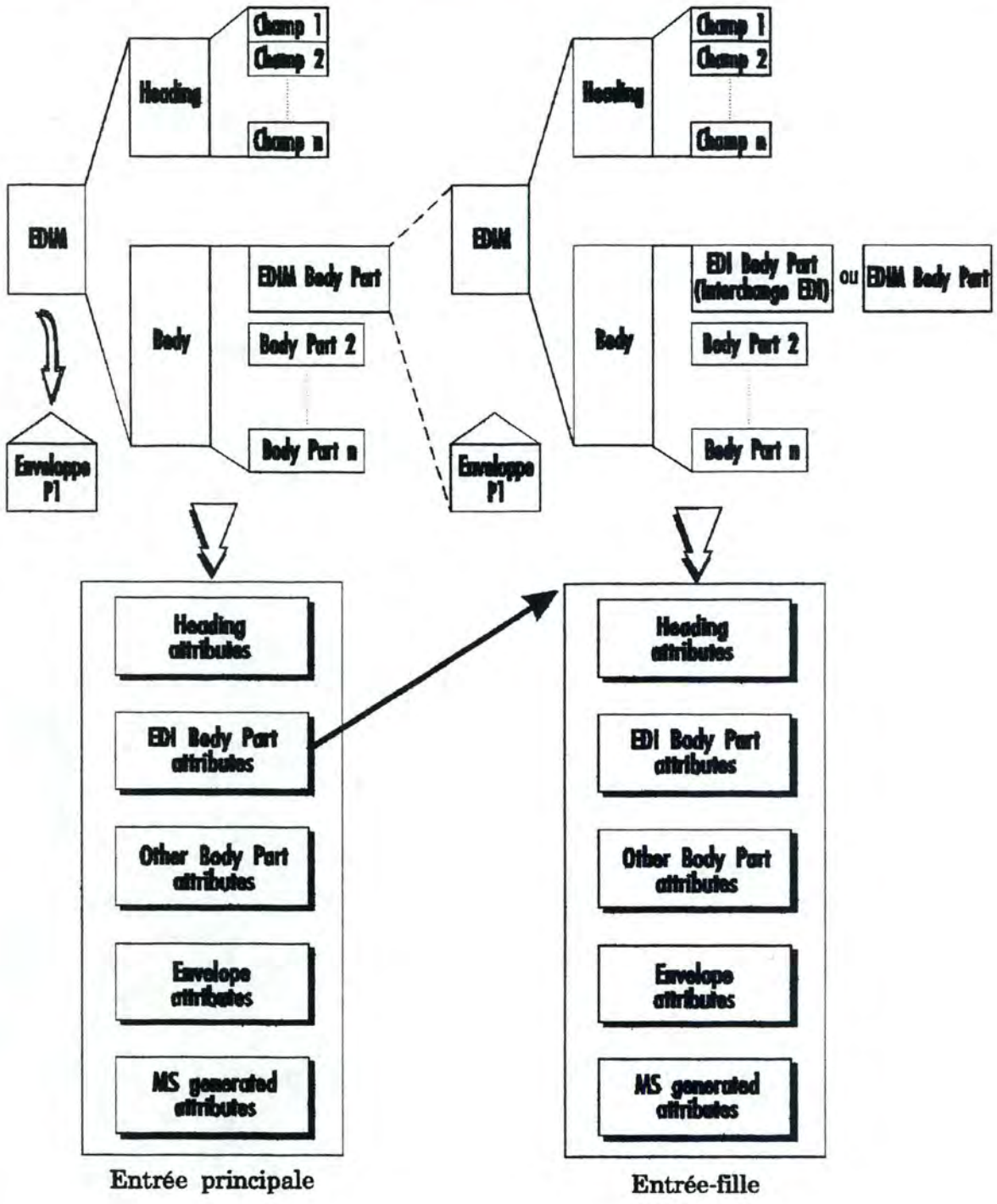

Figure 4-2 : Constitution des entrées MS pour un EDIM-relayé

certains messages, et d'avertir automatiquement (*auto-alert*) l'UA de l'arrivée de certains messages. Pour ces deux *auto-actions*, l'UA doit préalablement fournir au MS les critères de sélection.

De ces deux *auto-actions*, l'*auto-alert* peut être utilisée sans restriction pour l'EDI. L'*auto-forward* semble à première vue correspondre à merveille à la philosophie de X.435, car le relais (*forwarding*) de l'EDIM est un des principes de base du protocole Pedi. Mais CCITT estime que l' *auto-forward* telle qu'elle est définie dans X.400-88

ne convient pas bien à l'EDI, car des notions importantes telles que la responsabilité (voir 3.4) n'y sont pas présentes. Pour cette raison, CCITT a préféré créer spécialement une autre *auto-action* nommée *EDI auto-forward* pour l'EDI. Il y a deux sortes d' *EDI auto-forward*:

***EDI auto-forward* sans acceptation de responsabilité**

L'EDI-MS relaie automatiquement à un ou plusieurs autres récepteurs tout EDIM qui satisfait aux critères que l'utilisateur a préalablement définis. L'EDI-MS n'accepte pas la responsabilité, laisse l'EDIM reçu inchangé et envoie une notification de relais (FN) à l'émetteur si cela avait été demandé.

Cette situation pourrait arriver, par exemple, à une petite compagnie qui peut demander à l'EDI-MS d'un VAN (*Value Added Network*) de relayer avec renvoi de FN tout EDIM de type "INVOIC" (facture) à un service de comptabilité externe. Ce dernier acceptera la responsabilité et paiera la facture au nom de la petite compagnie. [HILL]

***EDI auto-forward* avec acceptation de responsabilité**

L'EDI-MS relaie automatiquement à un ou plusieurs autres récepteurs tout EDIM qui satisfait aux critères que l'utilisateur a préalablement définis. L'EDI-MS accepte la responsabilité, mais **doit laisser l'EDIM reçu inchangé** et envoie une notification positive (PN) à l'émetteur si cela avait été demandé.

Cette situation pourrait arriver, par exemple, à une petite compagnie qui peut demander à l'EDI-MS d'un VAN (*Value Added Network*) de relayer avec renvoi de PN tout EDIM de type "ORDERS" (commande) aussi bien à un EDI-UA fonctionnant au sein de la petite compagnie qu'à un service de comptabilité externe. [HILL]

Contrairement à un relais normal d'EDIM avec acceptation de responsabilité par un EDI-UA (voir 3.4), l' *EDI auto-forward* avec acceptation de responsabilité par un EDI-MS ne permet pas à ce dernier de modifier l'EDIM reçu en enlevant ou ajoutant des *body parts*.

4.1.3 Conséquences de l'absence de l'EDI-MS pour X.435/X.400-84

L'EDI-MS ne peut bien entendu être disponible dans une application X.435/X.400-84, car le concept du MS fut introduit seulement dans X.400-88. L'absence de l'EDI-MS prive de ce fait une application X.435/X.400-84 des possibilités intéressantes telles que la recherche sélective et l' *auto-action*.

Pourtant, l'absence de l'EDI-MS ne nuit pas à la conformité d'une application X.435/X.400-84 vis-à-vis de la Recommandation X.435, car cette dernière définit l'EDI-MS comme étant un concept **optionnel** et non obligatoire. Cela signifie qu'une

application X.435 (utilisant X.400-88) peut très bien ne pas prévoir l'usage de l'EDI-MS et rester malgré cela conforme. A fortiori, le même raisonnement vaut aussi pour une application X.435/X.400-84 où le concept du MS est totalement absent.

De plus, le problème de l'absence de l'EDI-MS peut partiellement être contourné par l'instauration d'un serveur central permanent pour éviter tout débordement des EDIMs en entrée. Ce sujet sera traité dans 5.4.2 lorsqu'on discutera de l'architecture du prototype X.435/X.400-84 que l'auteur de ce texte a développé.

L'absence de l'EDI-MS n'est donc pas un obstacle insurmontable pour X.435/X.400-84. Mais encore une fois, cela ne signifie pas que cette absence n'a pas de conséquence négative: au contraire, l'EDI-MS est un concept important et utile, et son absence est très regrettable.

4.2 Intégration de X.500 dans X.400-84

Avant d'envoyer un EDIM à travers le MTS X.400, l'émetteur doit d'abord obtenir l'*O/R Address* de l'EDI-UA du récepteur. Cela n'est pas un problème dans un environnement EDI à accord bilatéral, où l' *O/R Address* de chaque utilisateur EDI est échangé durant la négociation bilatérale et l'obtention de l' *O/R Address* peut être faite de façon locale par l'EDI-UA. Toutefois, dans un environnement EDI ouvert à tout le monde (*Open EDI*), où chaque EDI-UA peut émettre/recevoir des EDIMs à/de tout partenaire potentiel sans accord préalable, l'utilisation des services de répertoire X.500 (voir 2.6.3) deviendra nécessaire.

4.2.1 *Name Resolution*

Dans un environnement EDI, chaque utilisateur EDI est identifié par un *EDI Name*, qui est une chaîne de caractères alphanumérique et arbitraire. L' *EDI Name* doit être unique à l'intérieur d'une communauté particulière d'EDI. De grandes communautés EDI telles que les groupes CEFIC et EDIFICE utilisent des *EDI Names* délivrés par une autorité internationalement reconnue, par exemple DUNS pour ANSI X.12 et EAN pour EDIFACT. Un groupe commercial privé tel qu'une société multinationale ou un VAN peut définir librement ses *EDI Names* aussi longtemps que chaque *EDI Name* est unique au sein du groupe.

Dans le cas de X.435, un utilisateur EDI peut interroger X.500 et obtenir, à partir de l'*EDI Name* du récepteur de l'EDIM, l' *O/R Address* de l'EDI-UA de ce dernier. Ce processus est appelé *Name Resolution*. Pour le codage EDIFACT, l' *EDI Name* est transporté dans le segment UNB de l'interchange EDI (voir 1.2.6).

Voici les différentes étapes de la *Name Resolution* pour X.435, dans le cas où un nom EAN (EDIFACT) est utilisé comme *EDI Name*: [HILL]

A) L'application EDI passe le nom EAN à l'EDI-UA. Le nom EAN est "ABCDE".

B) L'EDI-UA appelle le DUA avec le Directory Name: *Country*=US, *Organization*=EAN, *EDIUser*=''ABCDE''.

C) Le DUA retourne le ''aliassed'' Directory Name: *Country*=US, *Organization*=TELECOMEDI, *OrganizationUnit*=SALES, *EDIUser*=INVOICES.

D) L'EDI-UA appelle le DUA avec le Directory Name retourné dans C afin d'obtenir l'*O/R Address* de l'EDI-UA du récepteur.

Il y a donc deux appels au DUA. Un premier (l'étape B) pour obtenir le Directory Name de l'EDI-UA du récepteur, et un deuxième pour obtenir l'O/R Address (voir figure 4-3).

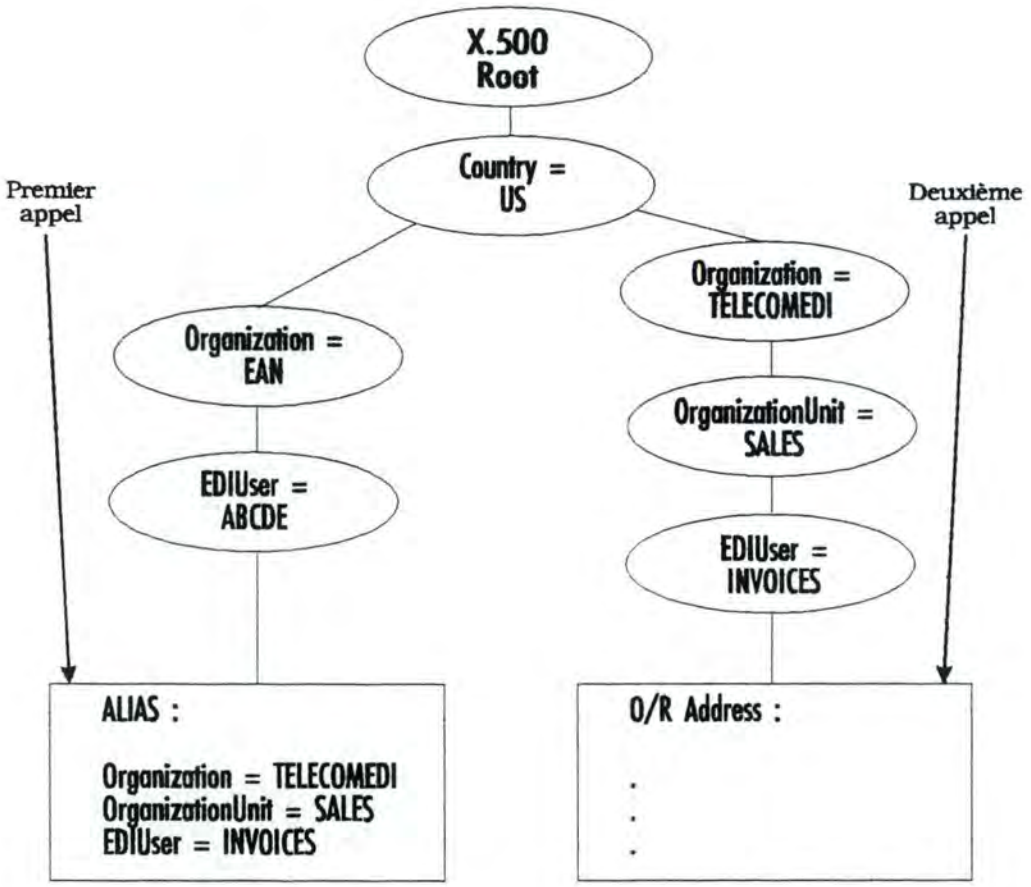

Figure 4-3 : Utilisation de X.500 pour la "Name Resolution" d'EDI Name

L'annexe J de X.435 spécifie que, après la dernière étape décrite ci-dessus, le DUA retourne non seulement l'*O/R Address* de l'EDI-UA du récepteur, mais aussi optionnellement ses attributs spécifiques à l'EDI. Citons quelques-uns de ces attributs:

EDIBodyPartType: Les codages EDI qui sont acceptés par l'application EDI réceptrice (ex.: EDIFACT, ANSI X.12, ...).

StandardVersion: La version de codage EDI qui est acceptée par l'application EDI réceptrice.

DocumentType: Les types d'interchange qui sont acceptés par l'application EDI réceptrice (ex.: facture, bon de commande, ...)

Cette possibilité est très intéressante. Cela signifie que, en effectuant des requêtes aux services de répertoire X.500, un émetteur peut apprendre si le récepteur sera capable d'accepter et de traiter l'interchange EDI qu'il voudrait lui envoyer. Le risque d'un EDIM refusé sera alors minimisé.

4.2.2 Difficulté éventuelle d'intégration de X.500 dans X.435/ X.400-84

Bien que l'*Open EDI* soit encore un rêve lointain, les services de répertoire X.500 peuvent certainement se révéler indispensables le jour venu. Mais puisque X.500 a été introduit seulement en novembre 1988, c'est-à-dire bien après l'apparition de X.400-84, on peut se demander s'il est possible de l'intégrer dans une application X.435/X.400-84.

Avant de répondre à cette question, il convient de comprendre le fonctionnement de la couche Application de l'OSI. Dans l'environnement OSI, la part du système qui exécute un traitement d'information pour une application particulière est appelé le processus Application. Le processus Application peut être divisé en deux parties: l'Agent Application et l'Entité Application. L'Agent Application est dépendant du système, il sert d'interface avec l'utilisateur et avec le système sur lequel il est installé, dans le but de fournir l'accès aux ressources locales du système. Quant à l'Entité Application, elle fait partie de la couche Application de l'OSI. Elle exécute des activités d'application qui sont indépendantes du système, et qui sont disponibles comme des éléments de service d'application à l'Agent Application. [HS]

Les éléments de services de l'Entité Application peuvent être classés en deux catégories:

SASE (Specific Application Service Elements) :
Eléments de service qui sont relatifs à la nature spécifique du standard en question: services de fichier pour FTAM, services de messagerie pour X.400, etc. [HS]

CASE (Common Application Service Elements) :
Eléments de service pouvant être utilisés communément par tous les standards: ACSE (Association Control Service Elements), RTSE (Reliable Transfer Service Elements), ROSE (Remote Operations Service Elements), etc.

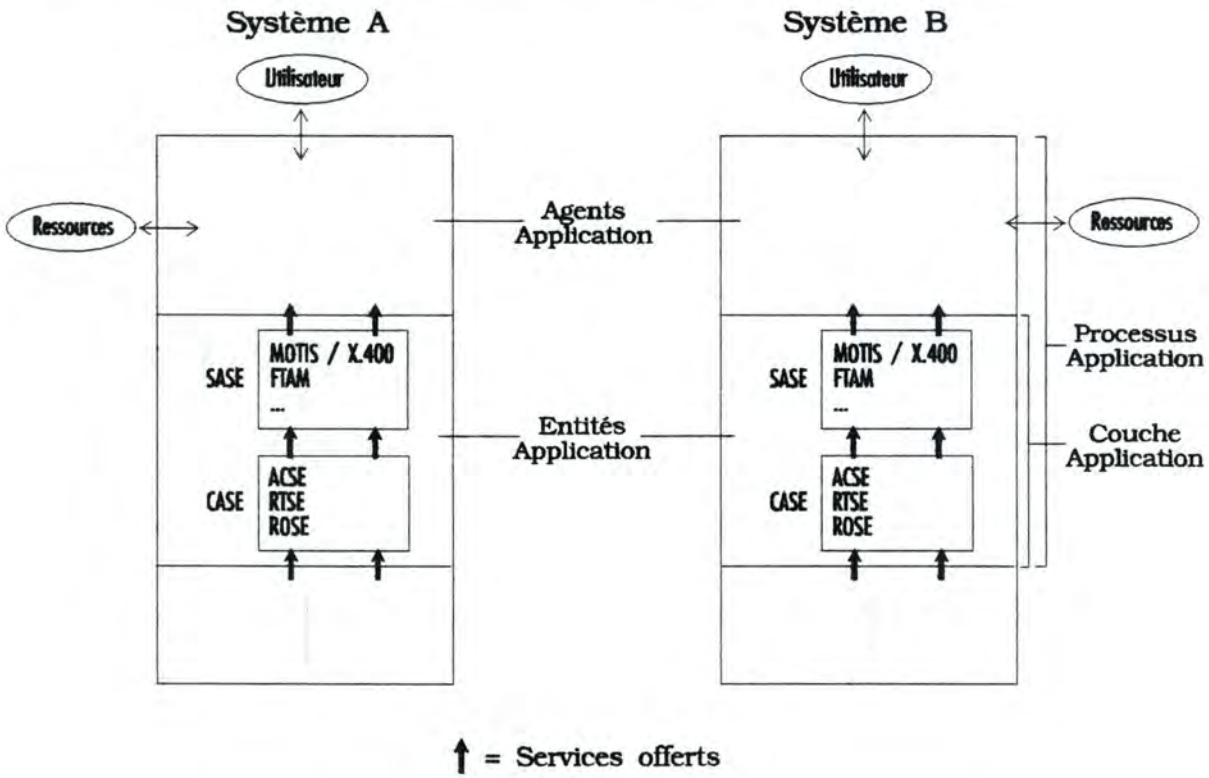

Figure 4-4 : Processus Application du modèle OSI

La figure 4-4 montre la disposition des tous ces composants du processus Application.

X.500 utilise deux des éléments de service CASE: ACSE et ROSE. Or, ni l'un ni l'autre n'est utilisé par X.400-84, car ACSE et ROSE n'ont pas encore été définis au moment où X.400-84 est apparu. ACSE et ROSE risquent donc de ne pas être disponibles dans un système sur lequel tourne X.400-84 (voir figure 4-5). Ainsi, pour implémenter un DUA de X.500 dans un tel système, nous avons deux solutions

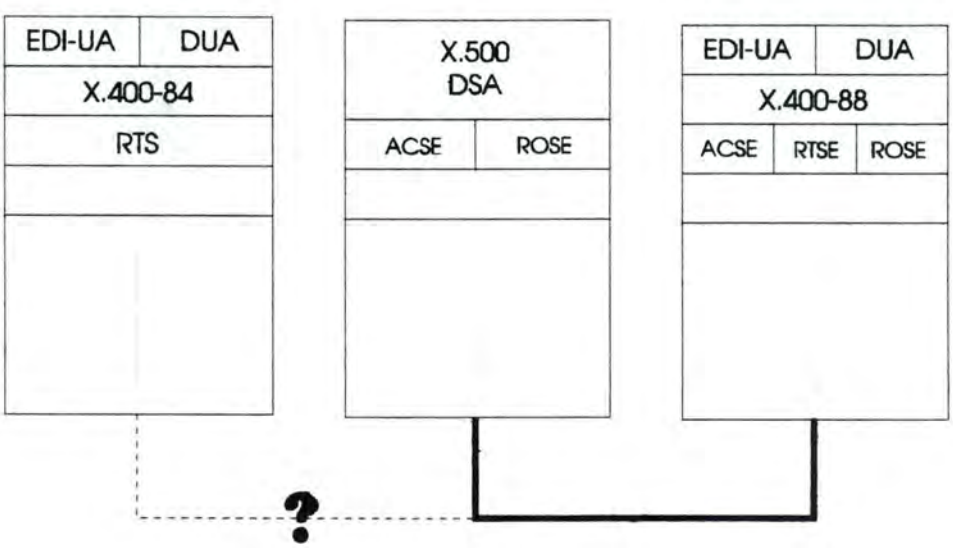

Figure 4-5 : Difficulté d'intégration de X.500 dans EDI / X.400-84

possibles:

1) Acquérir ACSE et ROSE. C'est certainement la solution la plus simple. D'ailleurs, des implémentations gratuites d'ACSE et de ROSE existent sur le marché (voir ISODE dans 5.3.3).

2) Modifier la couche Application. Ecrire des codes nécessaires pour l'Entité Application dans le but de fournir la part de fonctionnalités d'ACSE et de ROSE dont un DUA a besoin.

L'intégration de X.500 dans X.435/X.400-84 peut alors se faire sans aucun problème.

4.3 Absence des services de sécurité dans X.400-84

Nous l'avons déjà dit, les réseaux de télécommunication sont vulnérables à des actions mal intentionnées et illégales. Pour contrer de telles menaces, CCITT a défini dans X.400-88 des services de sécurité (voir 2.6.4), dont beaucoup sont utilisés par X.435. Pour X.435 ensuite, CCITT a introduit d'autres services de sécurité propres à l'EDI pour compléter ceux qui sont déjà définis dans X.400-88.

Tant les services de sécurité de X.400 que ceux de X.435 sont surtout basés sur l'utilisation des algorithmes asymétriques d'encryptage de données. Nous allons donc avoir une brève discussion sur le principe des ces algorithmes, discussion qui servira d'introduction pour la description des services de sécurité de X.435.

4.3.1 Algorithme asymétrique d'encryptage de données

En cryptographie, il existe deux sortes d'algorithme d'encryptage de données: l'algorithme symétrique à une seule clé secrète et l'algorithme asymétrique à deux clés, une secrète et une publique.

Dans le cas de l'algorithme symétrique, une seule clé secrète est utilisée par l'émetteur et le récepteur. L'émetteur utilise la clé pour encrypter des données et le récepteur utilise la même clé pour décrypter. La clé ne doit être connue par aucune autre personne, sinon il y a le risque qu'un intrus qui détiendrait la clé l'utilise pour se faire passer pour le propriétaire de la clé. Et même si le secret est bien gardé, puisqu'un émetteur A doit communiquer sa clé secrète à un récepteur B pour entrer en communication avec celui-ci, il y a toujours le risque que B divulgue cette clé ou utilise cette clé à l'insu de A pour se faire passer pour A et effectuer des actions illégales. On voit très vite le problème que cela pose à un MHS X.400, où pour garantir une parfaite sécurité, tout utilisateur devrait utiliser une clé différente pour chaque autre utilisateur avec qui il aimerait entrer en communication. Cela est tout simplement impraticable.

D'où l'intérêt de l'algorithme asymétrique, qui fonctionne de la manière suivante: l'émetteur utilise une clé secrète connue de lui seul pour encrypter des données et le

récepteur utilise une autre clé publique - c'est-à-dire connue de tout le monde - pour décrypter. Dans ce cas-ci, si la clé secrète est bien gardée, personne d'autre ne peut se faire passer pour son propriétaire, car seule la clé publique est connue et elle ne peut pas servir à encrypter. L'algorithme asymétrique d'encryptage de données est couramment utilisé pour X.400. Dans un MHS X.400, le récepteur d'un message peut par exemple faire appel aux services de répertoire X.500 pour connaître la clé publique de l'émetteur et décrypter les données envoyées par ce dernier.

Il est très important de remarquer que l'utilisation de la clé secrète pour l'encryptage et de la clé publique pour le décryptage a surtout pour but d'authentifier l'origine du message. Mais puisque la clé publique est connue de tout le monde, il existe bien entendu le risque d'interception et de décryptage illégal du message par un intrus. Dans le cas où l'on cherche plutôt à garantir la confidentialité du message, il faut inverser le processus: l'émetteur encrypte le message avec la clé publique **du récepteur**, et le récepteur décrypte le message avec sa clé secrète que lui seul connaît. Les deux méthodes peuvent bien sûr être combinées pour garantir l'origine **et** la confidentialité du message.

Beaucoup d'algorithmes asymétriques ont été inventés à ce jour. Les plus connus d'entre eux sont certainement l'algorithme RSA et l'algorithme Trapdoor-Knapsack. Il semblerait cependant que le CCITT ait une préférence pour l'algorithme RSA, car seul celui-ci est décrit dans une de ses Recommandations, la Recommandation X.509.

Algorithme RSA

L'algorithme RSA porte comme nom les initiales de ses inventeurs (Rivest - Shamir - Adleman). Il est basé sur le principe que la multiplication de deux nombres premiers (c'est-à-dire divisibles seulement par 1 et par eux-mêmes) est très simple à effectuer, alors que la factorisation du résultat d'une telle multiplication est extrêmement difficile à réaliser, surtout lorsque les deux nombres premiers de départ sont très grands. Nous allons voir très brièvement le principe du fonctionnement de cet algorithme extrêmement compliqué, les lecteurs peuvent trouver à l'Annexe C de la Recommandation X.509 une description plus complète.

A) Soient

p : grand nombre premier (p secret)
q : grand nombre premier (q secret)
r = p * q (r public)
SK : clé secrète (choisie au départ)
PK : clé publique (à calculer)
X : texte clair à encrypter

B) Totient d'Euler Ø() :

Ø(r) = (p-1) * (q-1)

C) Règle de congruence (Notation: ''mod'') :

Soient trois entiers a, b et c:
a = b mod c (a est congruent à (b mod c))
si l'on peut enlever un certain nombre de fois de c à a pour que le résultat soit égal à b. Autrement dit,
a modulo c = b modulo c

Exemple:
16 = 1 mod 5 (car 16 modulo 5 = 1 modulo 5 = 1)
23 = 9 mod 7 (car 23 modulo 7 = 9 modulo 7 = 2)

D) Si a - b mod c, on peut déduire

$a*n = b*n \bmod c$ (n quelconque)
et
$a^m = b^m \bmod c$ (m quelconque)

E) Propriété (admis sans démonstration) :

Pour tout entier a tel que a et r soient premiers entre eux:
$a^{\emptyset(r)} = 1 \bmod r$

F) A partir de D) et de E), on a

$a^{m*\emptyset(r)} = 1^m \bmod r$
$a^{m*\emptyset(r)} = 1 \bmod r$
$a^{m*\emptyset(r)+1} = a \bmod r$

G) Relation entre PK et SK :

On choisit au départ SK tel que SK et Ø(r) soient premiers entre eux.

La relation entre SK et PK est
$SK * PK = m * \emptyset(r) + 1$ (m entier)

H) Relation entre X encrypté et X

De F) et G), on a
$X^{SK*PK} = X^{m*\emptyset(r)+1} - X \bmod r$ (condition: X < r)

La formule qui lie le texte clair (X) et le texte encrypté (X^{SK*PK}) est donc
$X^{SK*PK} = X \bmod r$ (condition: X < r)

Résumons-nous: on choisit au départ deux grands nombres premiers p et q, et on calcule r puis Ø(r). On choisit ensuite SK tel que SK et Ø(r) soient premiers entre eux. On peut alors calculer la valeur de PK grâce à la relation de G.

Quand l'émetteur veut envoyer un texte X, il calcule le texte encrypté X^{SK} et l'envoie au récepteur. PK et r étant publics (pouvant être obtenus de X.500, par exemple), le récepteur utilise PK pour calculer X^{SK*PK} et, grâce à la formule de H, utilise r pour retrouver X. Le récepteur est sûr de la bonne origine de X, car seul l'émetteur est

capable de calculer X^{SK}.

Nous allons donner un exemple concret pour illustrer l'algorithme RSA. Pour faciliter les calculs, nous prenons de petites valeurs pour p et q:

L'émetteur décide de prendre les valeurs suivantes:
p = 47
q = 61
r = p * q = 2867
Ø(r) = (p-1) * (q-1) = 2760
SK = 167

Grâce à la relation de G, on calcule PK:
167 * PK = 1 + m * 2760
On obtient
PK = 1223 et m = 74 (la valeur de m est sans importance pour la suite)

L'émetteur décide d'envoyer un texte en encryptant deux octets à la fois. Par exemple, si les deux premiers octets ont les valeurs EBCDIC 18 et 19, alors il encrypte la valeur 1819:
X = 1819 (OK car 1819 < r = 2867)
X^{SK} modulo r = 1819^{167} modulo 2867 = 804
L'émetteur envoie 804 au récepteur.

Le récepteur reçoit la valeur 804 et essaie de retrouver X grâce à PK, r et la formule de H:
X^{SK*PK} modulo r = 804^{1223} modulo 2867 = 1819 = X

Le récepteur retrouve bien la valeur initiale 1819.

4.3.2 Sécurité dans X.435

X.435 utilise les services de sécurité X.400 vus dans 2.6.4. De plus, X.435 définit cinq services de sécurité supplémentaires spécifiques à l'EDI. Ces services sont:

- -Proof of EDI Notification
- -Non-repudiation of EDI Notification
- -Proof of content received
- -Non-repudiation of content received
- -Non-repudiation of content originated

La table 4-1 montre la liste complète des services de sécurité X.435, avec des indications sur le fournisseur et l'utilisateur de chacun de ces services.

De nouveaux éléments de service sont requis pour fournir ces services de sécurité. On trouve à l'Annexe I de [X.435] la liste complète des ces éléments de service qui doivent venir s'ajouter à ceux déjà disponibles dans X.400-88.

Services de sécurité	MTS-user émetteur	MTS	MTS-user récepteur
Proof of EDI Notification	U	-	P
Non-repudiation of EDI Notification	U	-	P
Proof of content received	U	-	P
Non-repudiation of content received	U	-	P
Non-repudiation of content originated	P	-	U

P = Fournisseur du service (Provider)
U = Utilisateur du service

Table 4-1 : Fourniture et utilisation des services de sécurité X.435 [X.435]

Pour fournir les services de sécurité listés ci-dessus, on utilise l'algorithme asymétrique d'encryptage de données. Ce dernier étant très lent, on utilise en général d'abord une fonction de hashing pour réduire la taille du message avant de l'encrypter. La fonction de hashing est un processus de calcul qui réduit un grand nombre de bits en un petit nombre de bits, de manière à ce que tous les bits originaux influencent le résultat du calcul et que deux messages en entrée différents n'aient pas le même résultat en sortie après le calcul. [HILL]

Nous allons voir comment grâce à l'algorithme asymétrique d'encryptage de données et la fonction de hashing, les cinq services de sécurité de X.435 peuvent être réalisés. Certaines explications qui suivent sont reprises du chapitre 10 de [HILL].

1) <u>Proof of EDI Notification</u>

Ce service fournit à l'émetteur d'un EDIM[1] la confirmation que l'EDIN qu'il reçoit était bien envoyée par le récepteur de l'EDIM.

Voici les différentes étapes de ce service:

Emetteur de l'EDIM:

A) Demander PN, NN et FN avec authentification (*proof*). Cela est fait en mettant le champ *edi-notification-security* du champ *edi-notification-requests-field* de l'EDIM à la valeur ''proof''. (Voir 8.2.3.3 dans [X.435] et 3.2.1 pour la signification de ces champs)

B) Mettre l'EDIM dans un message P1 et transmettre le tout au récepteur.

Récepteur de l'EDIM (Emetteur de l'EDIN):

C) Appliquer la fonction de hashing à l'EDIN. Encrypter le résultat du hashing avec l'algorithme asymétrique et la clé secrète du récepteur, mettre le résultat de l'encryptage dans le champ *content-integrity-check* (CIC) de l'enveloppe P1.

D) Mettre l'EDIN dans un message P1 et transmettre le tout à l'émetteur.

Emetteur de l'EDIM (Récepteur de l'EDIN):

E) Appliquer la fonction de hashing à l'EDIN reçue.

F) Décrypter le résultat encrypté de la fonction de hashing qui a été envoyé avec le message P1 reçu et qui se trouve dans le champ CIC (voir l'étape C). Le décryptage se fait en utilisant la clé publique du récepteur.

G) Comparer les résultats des étapes E et F. Si les résultats sont identiques, alors l'EDIN reçue a bien été envoyée par le récepteur, car seul le récepteur est capable d'encrypter le résultat de la fonction de hashing avec sa clé secrète.

La figure 4-6 illustre les étapes de ce service.

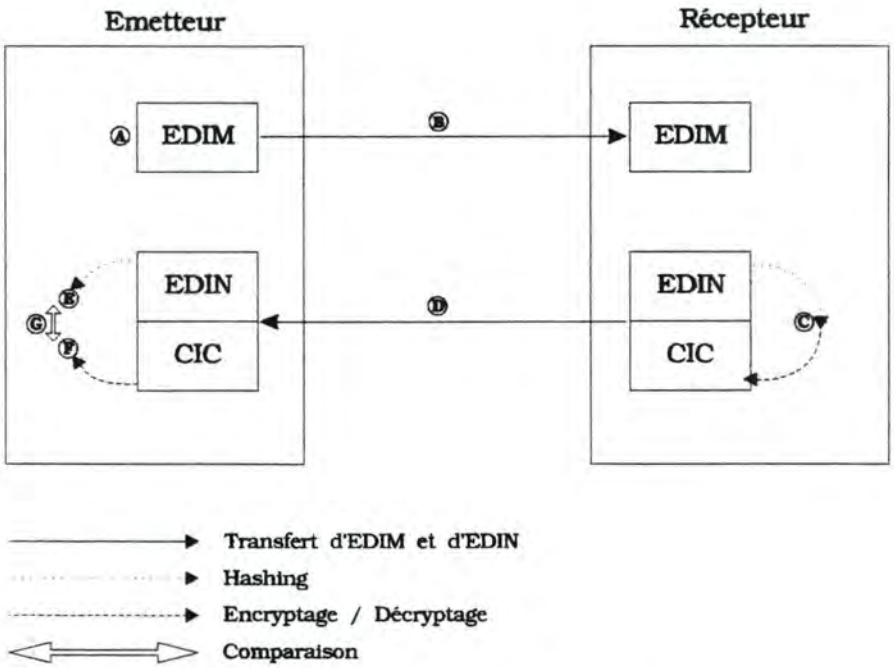

Figure 4-6 : Fonctionnement du service de sécurité "Proof of EDI Notification"

2) Non-repudiation of EDI Notification

Ce service fournit à l'émetteur d'un EDIM la preuve irréfutable que l'EDIN qu'il reçoit était bien envoyée par le récepteur de l'EDIM. Cette preuve protège l'émetteur de l'EDIM contre toute tentative du récepteur de l'EDIM de renier à tort l'envoi de l'EDIN.

Voici les différentes étapes de ce service:

Emetteur de l'EDIM:

A) Demander PN, NN et FN avec authentification non répudiable. Cela est fait en mettant le champ *edi-notification-security* du champ *edi-notification-requests-field* de l'EDIM à la valeur ''non-repudiation''.

B) Mettre l'EDIM dans un message P1 et transmettre le tout au récepteur.

Récepteur de l'EDIM (Emetteur de l'EDIN):

C) Appliquer la fonction de hashing à l'EDIN. Encrypter le résultat du hashing avec l'algorithme asymétrique et la clé secrète du récepteur, mettre le résultat de l'encryptage dans le champ *content-integrity-check* (CIC) de l'enveloppe P1.

C-bis) Obtenir un certificat d'origine non répudiable de la part d'une autorité compétente.

D) Mettre l'EDIN et le certificat dans un message P1 et transmettre le tout à l'émetteur.

Emetteur de l'EDIM (Récepteur de l'EDIN):

E) Appliquer la fonction de hashing à l'EDIN reçue.

F) Décrypter le résultat encrypté de la fonction de hashing qui a été envoyé avec le message P1 reçu et qui se trouve dans le champ CIC (voir l'étape C). Le décryptage se fait en utilisant la clé publique du récepteur.

G) Comparer les résultats des étapes E et F. Si les résultats sont identiques, alors l'EDIN reçue a bien été envoyée par le récepteur, car seul le récepteur est capable d'encrypter le résultat de la fonction de hashing avec sa clé secrète. La présence du certificat d'origine non répudiable (voir l'étape C-bis) en est la preuve irréfutable.

3) Proof of content received

Ce service fournit à l'émetteur d'un EDIM la confirmation que le contenu du message reçu par le récepteur était bien identique à celui qu'il avait envoyé.

Voici les différentes étapes de ce service:

Emetteur de l'EDIM:

A) Demander PN et NN avec authentification (*proof*). Cela est fait en mettant le champ *edi-notification-security* du champ *edi-notification-requests-field* de l'EDIM à la valeur ''proof''.

B) Demander PN et NN avec retour du message envoyé. Cela est fait en mettant

le champ *edi-reception-security* du champ *edi-notification-requests-field* de l'EDIM à la valeur ''proof''. (Voir 8.2.3.3 dans [X.435] et 3.2.1 pour la signification de ces champs)

C) Mettre l'EDIM dans un message P1 et transmettre le tout au récepteur.

Récepteur de l'EDIM (Emetteur de l'EDIN):

D) Mettre l'EDIM reçu dans le champ *original-content* du champ *notification-security-elements* de l'EDIN. (Conséquence de la demande de l'étape B)

E) Appliquer la fonction de hashing à l'EDIN. Encrypter le résultat du hashing avec l'algorithme asymétrique et la clé secrète du récepteur, mettre le résultat de l'encryptage dans le champ *content-integrity-check* (CIC) de l'enveloppe P1. (Conséquence de la demande de l'étape A)

F) Mettre l'EDIN dans un message P1 et transmettre le tout à l'émetteur.

Emetteur de l'EDIM (Récepteur de l'EDIN):

G) Appliquer la fonction de hashing à l'EDIN reçue.

H) Décrypter le résultat encrypté de la fonction de hashing qui a été envoyé avec le message P1 reçu et qui se trouve dans le champ CIC (voir l'étape E). Le décryptage se fait en utilisant la clé publique du récepteur.

I) Comparer les résultats des étapes G et H. Si les résultats sont identiques, alors l'EDIN reçue a bien été envoyée par le récepteur, car seul le récepteur est capable d'encrypter le résultat de la fonction de hashing avec sa clé secrète. (La demande de l'étape A est satisfaite)

J) Comparer le champ *original-content* du champ *notification-security-elements* de l'EDIN reçue (voir l'étape D) avec l'EDIM original. S'ils sont identiques, alors l'émetteur a la confirmation que l'EDIM que le récepteur a reçu était bien celui qu'il avait envoyé. (La demande de l'étape B est satisfaite)

La figure 4-7 illustre les étapes de ce service.

4) <u>Non-repudiation of content received</u>

Ce service fournit à l'émetteur d'un EDIM la preuve irréfutable que le contenu du message reçu par le récepteur était bien identique à celui qu'il avait envoyé. Cette preuve protège l'émetteur contre toute tentative du récepteur de renier à tort le contenu de l'EDIM reçu.

Voici les différentes étapes de ce service:

Emetteur de l'EDIM:

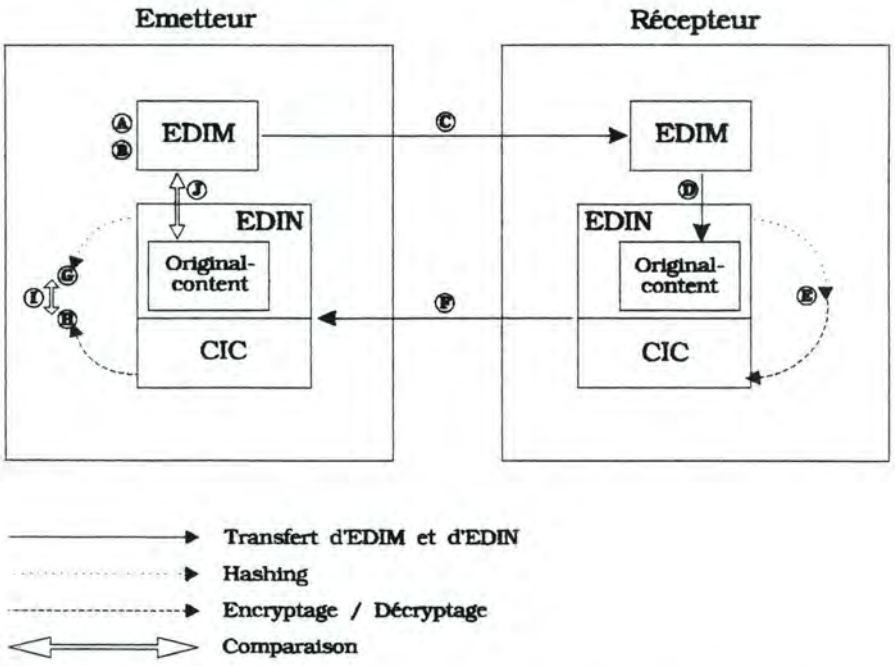

Figure 4-7 : Fonctionnement du service de sécurité "Proof of Content Received"

A) Demander PN et NN avec authentification non répudiable. Cela est fait en mettant le champ *edi-notification-security* du champ *edi-notification-requests-field* de l'EDIM à la valeur ''non-repudiation''.

B) Demander PN et NN avec retour du message envoyé. Cela est fait en mettant le champ *edi-reception-security* du champ *edi-notification-requests-field* de l'EDIM à la valeur ''non-repudiation''.

C) Mettre l'EDIM dans un message P1 et transmettre le tout au récepteur.

Récepteur de l'EDIM (Emetteur de l'EDIN):

D) Mettre l'EDIM reçu dans le champ *original-content* du champ *notification-security-elements* de l'EDIN. (Conséquence de la demande de l'étape B)

E) Appliquer la fonction de hashing à l'EDIN. Encrypter le résultat du hashing avec l'algorithme asymétrique et la clé secrète du récepteur, mettre le résultat de l'encryptage dans le champ *content-integrity-check* (CIC) de l'enveloppe P1. (Conséquence de la demande de l'étape A)

E-bis) Obtenir un certificat d'origine non répudiable de la part d'une autorité compétente.

F) Mettre l'EDIN et le certificat dans un message P1 et transmettre le tout à l'émetteur.

Emetteur de l'EDIM (Récepteur de l'EDIN):

G) Appliquer la fonction de hashing à l'EDIN reçue.

H) Décrypter le résultat encrypté de la fonction de hashing qui a été envoyé avec le message P1 reçu et qui se trouve dans le champ CIC (voir l'étape E). Le décryptage se fait en utilisant la clé publique du récepteur.

I) Comparer les résultats des étapes G et H. Si les résultats sont identiques, alors l'EDIN reçue a bien été envoyée par le récepteur, car seul le récepteur est capable d'encrypter le résultat de la fonction de hashing avec sa clé secrète. La présence du certificat d'origine non répudiable (voir l'étape E-bis) en est la preuve irréfutable. (La demande de l'étape A est satisfaite)

J) Comparer le champ *original-content* du champ *notification-security-elements* de l'EDIN reçue (voir l'étape D) avec l'EDIM original. S'ils sont identiques, alors l'émetteur a la confirmation que l'EDIM que le récepteur a reçu était bien celui qu'il avait envoyé. La présence du certificat d'origine non répudiable (voir l'étape E-bis) en est la preuve irréfutable. (La demande de l'étape B est satisfaite)

5) Non-repudiation of content originated

Ce service fournit au récepteur d'un EDIM la preuve irréfutable que le contenu du message reçu est bien identique à celui que l'émetteur avait envoyé. Cette preuve protège le récepteur contre toute tentative de l'émetteur de renier à tort le contenu de l'EDIM envoyé.

Voici les différentes étapes de ce service:

Emetteur de l'EDIM:

A) Appliquer la fonction de hashing à l'EDIM. Encrypter le résultat du hashing avec l'algorithme asymétrique et la clé secrète de l'émetteur, mettre le résultat de l'encryptage dans le champ *content-integrity-check* (CIC) de l'enveloppe P1.

B) Obtenir un certificat d'origine non répudiable de la part d'une autorité compétente.

C) Mettre l'EDIM et le certificat dans un message P1 et transmettre le tout au récepteur.

Récepteur de l'EDIM:

D) Appliquer la fonction de hashing à l'EDIM reçu.

E) Décrypter le résultat encrypté de la fonction de hashing qui a été envoyé avec le message P1 reçu et qui se trouve dans le champ CIC (voir l'étape A). Le

décryptage se fait en utilisant la clé publique de l'émetteur.

F) Comparer les résultats des étapes D et E. Si les résultats sont identiques, alors l'EDIM reçu est bien identique à celui envoyé par l'émetteur et a bien été envoyé par l'émetteur, car seul l'émetteur est capable d'encrypter le résultat de la fonction de hashing avec sa clé secrète. La présence du certificat d'origine non répudiable (voir l'étape B) en est la preuve irréfutable.

Les lecteurs attentifs auront remarqué que le récepteur n'a aucun moyen d'exiger l'envoi du certificat non répudiable et la présence du champ CIC, car l'envoi d'un EDIM est une initiative de l'émetteur. On peut cependant imaginer le récepteur refuser un EDIM important (une commande importante, par exemple) qui ne permet pas une non-répudiation du contenu envoyé et renvoyer à l'émetteur une notification négative (NN) avec comme code de raison *security-error* et comme code de diagnostic *repudiation-failure.*

4.3.3 Sécurité de bout en bout (*end-to-end*) dans X.435

Tous les services de sécurité discutés jusqu'à présent, que ce soient ceux définis dans X.400 ou ceux définis dans X.435, couvrent seulement l'EDI Messaging System (ou EDIMS, voir 3.1) et non l'EDI Messaging Environment (ou EDIME, voir 3.1). Ces services ne peuvent fonctionner qu'en supposant que les frontières entre l'EDIMS et le monde extérieur sont sûres et protégées, ce qui n'est pas toujours le cas. Pour mieux comprendre ce problème, regardons d'abord la figure 4-8.

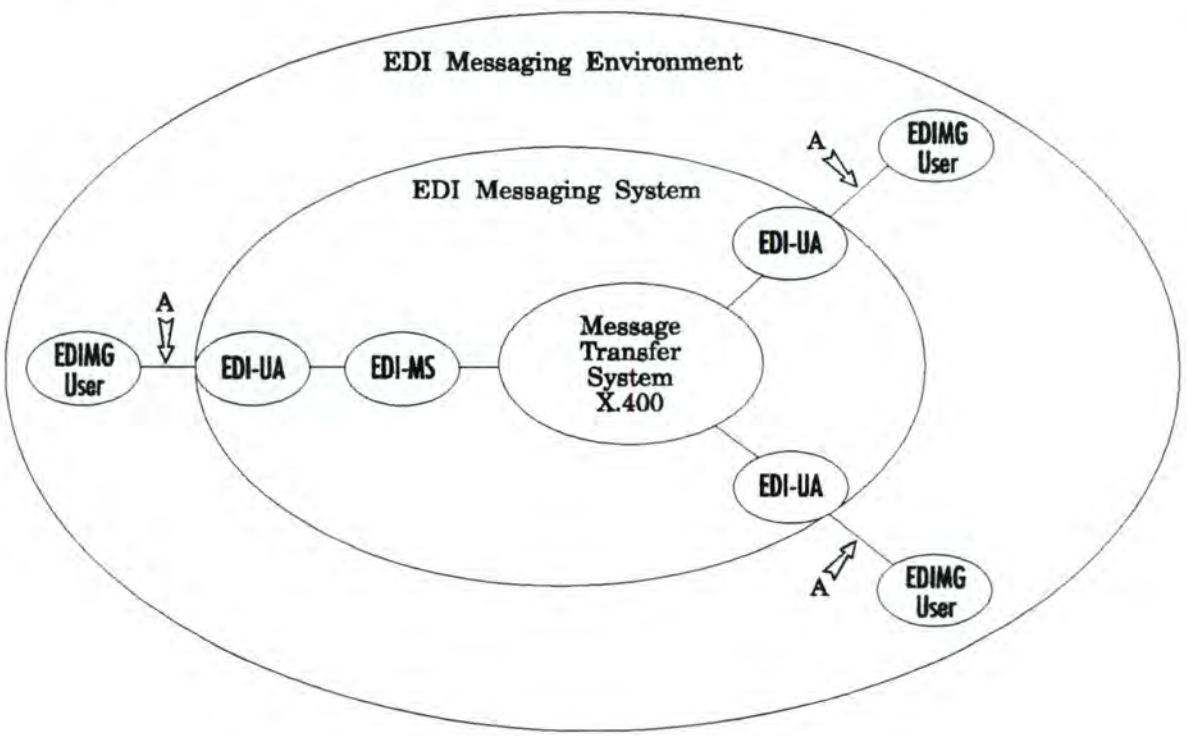

Fig. 4-8 : Attaques sur les frontières entre les EDIMG-users et l'EDIMS

Les services de sécurité de X.400 et de X.435 garantissent la sécurité entre les différents EDI-UAs et peuvent contrer d'éventuelles attaques qui surviennent à l'intérieur de l'EDIMS. Mais qu'arrive-t-il si un intrus décide d'attaquer "en amont", c'est-à-dire sur les frontières entre un EDI-UA et son EDIMG-user comme le montrent les points A de la figure 4-8? Autrement dit, un intrus pourrait par exemple intercepter au point A un interchange EDI envoyé par une application EDI, modifier le contenu, puis confier l'interchange falsifié à l'EDI-UA. Les services de sécurité discutés jusqu'à présent ne serviront alors à rien, car ils sont conçus pour protéger le contenu du message P1 (l'EDIM et l'EDIN en ce qui nous concerne) et ce genre d'attaque est tout simplement hors de leur zone de compétence.

Il faut donc trouver un moyen pour protéger le contenu de l'EDIM, c'est-à-dire l'interchange EDI. Autrement dit, assurer la sécurité entre les différents EDIMG-users. On voit bien qu'on se trouve devant un problème de la même nature que le problème de sécurité dans un MHS X.400, mais à un niveau supérieur.

X.435 prévoit la possibilité de contrer ce genre d'attaque. Pour cela, il existe des champs de données qui peuvent être utilisés pour fournir une sécurité de bout en bout (*end-to-end*) entre les EDIMG-users. Ces champs sont:

-Le champ *edi-application-security-elements* du *heading* de l'EDIM.

-Le champ *edi-application-security-elements* du champ *notification-security-elements* des *common-fields* de l'EDIN.

Ces champs ont la même fonction que le champ *content-integrity-check* (CIC) de l'enveloppe P1 et peuvent être utilisés de manière similaire. Alors que le champ CIC sert à contenir le résultat de l'encryptage de l'EDIM/EDIN, ces champs servent à contenir le résultat de l'encryptage de l'interchange EDI. Si on ne peut pas avoir confiance en les frontières entre l'EDI-UA et l'EDIMG-user, alors le rôle de ces champs devient très important.

Théoriquement, le principe de fonctionnement de tous les services de sécurité X.400 / X.435 faisant l'usage du champ CIC (voir par exemple 4.3.2) peut s'appliquer de manière similaire à ces champs pour fournir une sécurité de bout en bout. **Malheureusement, X.435 ne spécifie pas comment utiliser ces champs** . On ne sait pas non plus si l'important concept du certificat non répudiable sera utilisé à ce niveau. On a ainsi l'impression que CCITT nous montre la porte de secours sans nous donner la clef. Bien sûr, par un accord bilatéral, des partenaires EDI peuvent toujours définir pour eux l'utilisation de ces champs. **Mais faute de standard plus précis, la sécurité de bout en bout de X.435 est hélas pour le moment impossible pour de l'EDI ouvert à tout le monde (*Open EDI*).**

Nous allons donner deux exemples (repris de [HILL]) illustrant l'utilisation possible de chacun de ces deux champs. Bien sûr, cela suppose que l'émetteur et le récepteur de l'interchange EDI se sont préalablement mis d'accord sur la manière de les utiliser et sur l'algorithme à employer. Les lecteurs remarqueront que le principe de fonctionnement est similaire à celui décrit dans 4.3.2.

1) EDI Interchange origin authentication

Ce service fournirait à l'application EDI réceptrice d'un interchange EDI la confirmation que l'interchange EDI reçu a bien été envoyé par l'application EDI émettrice.

Voici les différentes étapes de ce service:

Application EDI émettrice:

A) Appliquer la fonction de hashing à l'interchange EDI. Encrypter le résultat du hashing avec l'algorithme asymétrique et la clé secrète de l'application EDI émettrice, demander à l'EDI-UA de mettre le résultat de l'encryptage dans le champ *edi-application-security-elements* du *heading* de l'EDIM.

B) Demander à l'EDI-UA d'envoyer l'EDIM à l'EDI-UA l'application EDI réceptrice.

Application EDI réceptrice:

C) Appliquer la fonction de hashing à l'interchange EDI reçu.

D) Obtenir de l'EDI-UA le résultat encrypté de la fonction de hashing qui a été envoyé avec l'EDIM reçu et qui se trouve dans le champ *edi-application-security-elements* du *heading* de l'EDIM (voir l'étape A). Le décrypter en utilisant la clé publique de l'application EDI émettrice.

E) Comparer les résultats des étapes C et D. Si les résultats sont identiques, alors l'interchange EDI reçu a bien été envoyé par l'application EDI émettrice, car seule cette dernière est capable d'encrypter le résultat de la fonction de hashing avec sa clé secrète.

2) Proof of EDIM content (EDI Interchange) received

Ce service fournirait à l'application EDI émettrice d'un interchange EDI la confirmation que l'interchange EDI reçu par l'application EDI réceptrice était bien identique à celui qu'il avait envoyé.

Voici les différentes étapes de ce service:

Application EDI émettrice:

A) Appliquer la fonction de hashing à l'interchange EDI. Sauvegarder le résultat pour la suite.

B) Demander à l'EDI-UA de demander la notification PN.

C) Demander à l'EDI-UA d'envoyer l'EDIM à l'EDI-UA de l'application EDI réceptrice.

Application EDI réceptrice:

D) Appliquer la fonction de hashing à l'interchange EDI reçu. Encrypter le résultat du hashing avec l'algorithme asymétrique et la clé secrète de l'application EDI réceptrice, demander à l'EDI-UA de mettre le résultat de l'encryptage dans le champ *edi-application-security-elements* du champ *notification-security-elements* des *common-fields* de l'EDIN.

E) Demander à l'EDI-UA d'envoyer l'EDIN à l'EDI-UA de l'application EDI émettrice.

Application EDI émettrice:

F) Obtenir de l'EDI-UA le résultat encrypté de la fonction de hashing qui a été envoyé avec l'EDIN reçue et qui se trouve dans le champ *edi-application-security-elements* du champ *notification-security-elements* des *common-fields* de l'EDIN (voir l'étape D). Le décrypter en utilisant la clé publique de l'application EDI réceptrice.

G) Comparer les résultats des étapes A et F. S'ils sont identiques, alors l'application EDI émettrice a la confirmation que l'interchange EDI que l'application EDI réceptrice a reçu était bien celui qu'il avait envoyé, car seule l'application EDI réceptrice est capable d'encrypter le résultat de la fonction de hashing avec sa clé secrète.

4.3.4 Conséquences de l'absence des services de sécurité pour X.435/X.400-84

L'absence des services de sécurité dans X.400-84 est un grand problème pour X.435/ X.400-84. Sans les services de sécurité, l'utilisateur d'une application X.435/X.400-84 ne peut jamais demander des preuves ou des non-répudiations pour les échanges EDI. Autrement dit, l'utilisateur n'est jamais certain de la bonne origine et de l'intégrité de l'EDIM/EDIN qu'il reçoit. C'est la porte ouverte aux intrus qui peuvent avoir tout le loisir d'intercepter et de modifier les EDIMs dans l'intention de nuire.

Ceci n'est pas seulement un problème pratique, mais également un problème juridique. Plusieurs pays de la Communauté Economique Européenne admettent ou sont sur le point d'admettre la légalité des échanges EDI par X.435. Mais sans la possibilité d'avoir des preuves ou des non-répudiations des échanges EDI, que se passerait-il lorsqu'il y a litige entre deux partenaires commerciaux? Sans les services de sécurité, il est tout simplement impossible d'authentifier les signatures digitales sur lesquelles se basent les législations concernant l'EDI.

On aurait pu espérer que la possibilité d'avoir une sécurité de bout en bout dans un

environnement EDI (voir 4.3.3) pourra pallier le manque des services de sécurité dans X.400-84 et résoudre en partie ce problème. En effet, avec cette possibilité, on aurait presque pu se passer des services de sécurité de X.400 et de X.435, car l'origine et l'intégrité de l'interchange EDI seront toujours garanties et un intrus ne peut qu'attaquer le *heading* et les *additional body parts* du message Pedi, ce qui est d'une moins grande gravité. Hélas, dans sa forme actuelle, la sécurité de bout en bout de X.435 est encore mal définie et ne permet qu'un usage restreint.

L'absence des services de sécurité est véritablement le tendon d'Achille de X.435/ X.400-84 et constitue le seul point sur lequel on peut avoir de sérieuses réserves.

4.4 Utilisation des MTAs version 1984 pour X.435

X.435 spécifie dans le paragraphe 19 les actions spéciales qui doivent être prises lorsqu'on soumet un message Pedi à un MTA version 1984 et lorsqu'un MTA version 1988 relaie un message Pedi à un MTA version 1984. Ces actions concernent les champs suivants de l'enveloppe P1:

-*content-type* (Type de contenu)

-*original-encoded-information-types* (Types originaux de l'information encodée)

4.4.1 Type de contenu

Le champ *content-type* indique le type de contenu qui se trouve dans le message P1. Le type de contenu le plus courant est certainement ''Message Interpersonnel'' (*Interpersonal Message*). Pour un EDIM, ce champ doit être mis à la valeur d'identifiant d'objet (*object identifier*) *''id-mct-pedi''*, ce qui correspond à la séquence d'entiers {2 6 7 10 0}.

L'EDI est normalement inconnu à X.400-84 et un MTA version 1984 est incapable de traiter un message P1 avec une telle valeur de type de contenu. Pour cette raison, X.435 spécifie les règles suivantes [X.435]:

-Quand un EDI-UA soumet un EDIM/EDIN à un MTA version 1984, il doit mettre le champ *content-type* à la valeur d'entier 35.

-Quand un MTA version 1988 relaie un EDIM/EDIN à un MTA version 1984, il doit remplacer la valeur d'identifiant d'object *''id-mct-pedi''* par la valeur d'entier 35.

Bien sûr, il faut que l'EDI-UA accepte aussi bien les messages dont le champ *content-type* est mis à la valeur *''id-mct-pedi''* que ceux dont ce champ est mis à la valeur 35.

La valeur d'entier 35 imposée par X.435 n'apparaît nulle part dans [X.400-84]. La raison de cette valeur reste un mystère pour nous.

4.4.2 Types originaux de l'information encodée

Le champ *original-encoded-information-types* (EITs) indique les types originaux de l'information encodée qui se trouve dans le message P1. Pour un EDIM, ce champ peut être mis à la valeur EDIFACT, ANSI X.12 ou autres codages EDI.

Encore une fois, l'EDI est normalement inconnu à X.400-84 et un MTA version 1984 est incapable de traiter un message P1 avec de tels EITs. Pour cette raison, X.435 spécifie les règles suivantes [X.435]:

> -Quand un EDI-UA soumet un EDIM à un MTA version 1984, il doit mettre à la valeur ''vrai'' le bit *undefined* du champ *original-encoded-information-types*, et doit laisser le champ *external-encoded-information-type* vide. (Voir 8.2.1.1.1.33 de la Recommandation X.411 pour la signification de ces champs)
>
> -Quand un MTA version 1988 relaie un EDIM à un MTA version 1984, il doit mettre à la valeur ''vrai'' le bit *undefined* du champ *original-encoded-information-types*, et doit laisser le champ *external-encoded-information-type* vide.

Par conséquent, lorsqu'un EDIM est soumis/relayé à/par un MTA version 1984, le MTA final (celui qui délivre l'EDIM à l'EDI-UA récepteur) ne sera pas capable de reconnaître les EITs de l'EDIM qu'il reçoit, car mis à la valeur '' *undefined*''. Il délivrera alors tous les EDIMs de ce type à son EDI-UA, même si ce dernier avait demandé que les EDIMs de certains EITs (par exemple ANSI X.12) ne lui soient pas délivrés. De même, si le MTA final et l'EDI-UA récepteur sont reliés à un EDI-MS, les recherches sélectives et les *auto-actions* basées sur la valeur d'EITs seront sans effet.

4.4.3 Conséquences de l'utilisation des MTAs version 1984 pour X.435/X.400-84

Les règles vues dans 4.4.1 et 4.4.2 sont très surprenantes. X.435 les présente comme des actions qui rendent une enveloppe P1 de la version 1988 compatible avec celle de la version 1984. X.435 n'en tire aucune conclusion et ne fait pas d'autres commentaires. Pourtant, ces règles apparemment anodines attirent toute notre attention sur le but de notre analyse: la faisabilité de l'utilisation de X.400-84 pour X.435.

Puisqu'un message Pedi peut être soumis à un MTA version 1984 **et** relayé par un MTA version 1984, **il est clair que CCITT n'exclut pas la possibilité d'utiliser X.400-84 pour X.435**! CCITT n'affirme pas cela explicitement et il est dommage que CCITT n'éclaircisse pas sa position à ce sujet.

Nous avons qualifié ces règles de surprenantes, car nous savons que lorsque qu'un

message P1 est soumis/relayé à/par un MTA version 1984, il perd automatiquement la protection des éventuels services de sécurité que son émetteur aurait requis (voir l'Annexe B de la Recommandation X.419). Un intrus peut alors attaquer le message après le passage de celui-ci à travers un tel MTA. Nous avons déjà vu dans 4.3.4 les sérieux problèmes que cela peut poser.

4.5 Eléments de service MT manquants de X.400-84 pour X.435

Les éléments de service MT exigés par X.435 sont bien entendu ceux de X.400-88, et beaucoup parmi eux ne sont pas fournis par X.400-84. Nous allons donc donner la liste complète des éléments de service MT manquants de X.400-84 pour X.435, en expliquant pour chacun d'entre eux les conséquences qu'il pourrait y avoir sur une implémentation X.435/X.400-84.

Pour ce faire, nous nous basons sur les trois documents suivants:

-Eléments de service de messagerie EDI (Paragraphe 14 et table 6 de [F.435]).

-Eléments de service de transfert de message version 1984 (Paragraphe 4 et table 1 de [X.400-84]).

-Eléments de service de transfert de message version 1988 (Paragraphe 19 et tables 4 & 5 de [X.400-88]).

Voici comment nous procédons: nous trouvons dans [F.435] tous les éléments de service MT exigés par X.435, éléments que nous comparons avec ceux effectivement disponibles dans [X.400-84]. Nous obtenons ainsi une liste d'éléments de service manquants de X.400-84 pour X.435. Nous pouvons dès lors trouver dans [X.400-88] les explications sur chacun des éléments de service de cette liste.

Cette liste comporte pas moins de 48 éléments de service qui manquent à X.400-84 pour répondre aux exigences de X.435. Leur nombre est très impressionnant à première vue, mais nous allons voir que les conséquences ne sont pas aussi graves qu'on aurait pu craindre.

Ces éléments de service peuvent être groupés en cinq catégories: ceux relatifs au *Message Store*, à la sécurité, au PDAU, à la liste de distribution, et les autres. Voici la liste complète des éléments de service manquants de X.400-84 pour X.435:

Message Store

Le problème concernant l'absence du *Message Store* a déjà été abordé dans 4.1. Nous n'allons donc pas revenir sur ce sujet et nous allons simplement citer ici les éléments de service qui concernent le MS:

1) **Stored Message Deletion**
2) **Stored Message Fetching**

3) **Stored Message Listing**
4) **Stored Message Summary**
5) **Stored Message Alert**
6) **Stored Message Auto-forward** (Son utilisation est déconseillée par X.435: voir 4.1.2)

Sécurité

Les éléments de service de sécurité de X.400-88 ont déjà été discutés dans 2.6.4 et le problème dû à leur absence a été abordé dans 4.1. Nous n'allons donc pas revenir sur ce sujet et nous allons simplement citer ici ces éléments de service:

7) **Content Confidentiality**
8) **Content Integrity**
9) **Message Flow Confidentiality**
10) **Message Origin Authentication**
11) **Message Security Labelling**
12) **Message Sequence Integrity**
13) **Non-repudiation of Delivery**
14) **Non-repudiation of Origin**
15) **Non-repudiation of Submission**
16) **Probe Origin Authentication**
17) **Proof of Delivery**
18) **Proof of Submission**
19) **Report Origin Authentication**
20) **Secure Access Management**

PDAU

Le PDAU a été discuté dans 2.6.2. Bien que son utilisation soit prévue dans X.435, son utilité paraît discutable dans un système de messagerie EDI. D'abord, le PDAU est généralement connecté à un service postal et délivre les EDIMs sous une forme physique (ex.: sur papier), ce qui est contraire au principe de l'EDI et réduit les avantages procurés par l'EDI à néant. Ensuite, le PDAU est incapable de soumettre des EDIMs au MTS X.400. C'est donc un composant qui est peu utile dans un système de messagerie EDI et qui ne sert qu'à recevoir occasionnellement quelques EDIMs sur du papier. Mais alors, on peut très bien s'en passer et utiliser s'il le faut les moyens classiques d'échange de documents (par exemple le service postal).

Pour ces raisons, nous n'allons pas décrire en détail les éléments de service de cette catégorie et nous allons simplement nous contenter de les citer:

21) **Additional Physical Rendition**
22) **Basic Physical Rendition**
23) **Counter Collection**
24) **Counter Collection with Advice**
25) **Delivery via Bureaufax Service**

26) **EMS (Express Mail Service)**
27) **Ordinary Mail**
28) **Physical Delivery Notification by MHS**
29) **Physical Delivery Notification by PDS**
30) **Physical Forwarding Allowed**
31) **Physical Forwarding Prohibited**
32) **Registered Mail**
33) **Registered Mail to Addressee in Person**
34) **Request for Forwarding Address**
35) **Special Delivery**
36) **Undeliverable Mail with Return of Physical Message**

Liste de distribution

La liste de distribution a été discutée dans 2.5.1. C'est un concept introduit dans X.400-88 et qui n'existe pas dans X.400-84. Voici les éléments de service relatifs à la liste de distribution:

37) **Use of Distribution**

''Elément de service qui permet à l'UA émetteur de spécifier une liste de distribution comme récepteur.''

La seule solution pour l'EDI-UA de X.435/X.400-84 de compenser le manque de cet élément de service est simplement de spécifier individuellement les *O/R Names* de tous les récepteurs. Or, il faut se rappeler que l' *O/R Name* d'une liste de distribution et l' *O/R Name* ''normal'' ont des apparences semblables et qu'un EDI-UA est incapable de les distinguer. Mais grâce à X.500, ce problème est facilement résolu: si on ne sait pas si un *O/R Name* particulier est une liste de distribution, on peut interroger X.500 avec l' *O/R Name* en question. Si effectivement il s'agit d'une liste de distribution, X.500 retournera tous les *O/R Address* de la liste, ce que l'EDI-UA utilisera pour spécifier les récepteurs.

38) **DL Expansion History Indication**

''Elément de service qui fournit à l'UA du récepteur des informations concernant la liste de distribution à travers laquelle le message est arrivé.''

La solution pour pallier le manque de cet élément de service est simple: l'EDI-UA récepteur peut interroger X.500 avec l' *O/R Name* qui se trouve dans le champ ''Originator'' du *heading* de l'EDIM reçu. S'il s'agit effectivement d'une liste de distribution, alors X.500 lui retournera tous les *O/R Address* de la liste.

39) **DL Expansion Prohibited**

''Elément de service qui permet à l'UA émetteur de demander au MTS X.400 de ne pas effectuer une expansion si l'O/R Name du récepteur est une liste de distribution. Dans ce cas-ci, le MTS renvoie un rapport de non-livraison à l'UA émetteur.''

Cet élément de service est utilisé par l'UA lorsque celui-ci ne désire pas que le message soumis soit reçu par plus d'un récepteur. Dans X.435, cet élément de service peut être utilisé lors de l'envoi d'un EDIM avec demande d'EDINs; il évite en effet à l'EDI-UA émetteur la réception d'un nombre imprévu d'EDINs à cause de l'expansion de la liste de distribution.

Encore une fois, l'EDI-UA émetteur peut toujours interroger X.500 avant de prendre le risque d'utiliser sans le vouloir une liste de distribution pour l'envoi d'un EDIM. En prenant cette précaution, cet élément de service devient inutile dans X.435/X.400-84.

Divers

Les éléments de service qui ne font pas partie des quatre autres catégories sont repris ici. Ils sont de nature très diverse et nous allons donner des explications sur chacun d'entre eux, avec des propositions de solution pour pallier leur manque dans X.435/ X.400-84 par des actions de l'EDI-UA.

40) **Conversion Prohibited in Case of Loss of Information**

"Elément de service qui permet à l'UA émetteur d'informer le MTS que la conversion des types d'information encodée (EITs) ne devrait pas être réalisée pour le message soumis si une telle conversion peut causer une perte d'information." (Voir X.408 pour la discussion sur la perte d'information)

La solution pour compenser le manque de cet élément de service est d'utiliser un autre élément de service plus radical et disponible dans X.400-84: "Conversion Prohibited". Ce dernier permet à l'UA émetteur de demander au MTS de n'effectuer la conversion d'EITs en aucun cas.

41) **Designation of Recipient by Directory Name**

"Elément de service qui permet à l'UA émetteur d'utiliser un *Directory Name* à la place d'un *O/R Address*."

Nous avons déjà vu dans 4.2 que l'intégration de X.500 dans X.400-84 ne pose pas de problème. Dès lors, au lieu de confier directement le *Directory Name* au MTS comme c'est permis dans X.400-88, l'utilisateur de X.435/X.400-84 peut d'abord effectuer la *Name Resolution* (voir 4.2.1), obtenir l' *O/R Address* de la part de X.500, puis donner celle-ci au MTS comme identifiant du récepteur. L'absence de cet élément de service peut ainsi être contournée.

42) **Latest Delivery Designation**

"Elément de service qui permet à l'UA émetteur de spécifier le moment au plus tard auquel le message doit être délivré. Si le MTS ne peut pas délivrer avant la date et l'heure spécifiées, alors le message n'est pas délivré et est rejeté."

Voici notre raisonnement pour contourner le manque de cet élément de service dans EDI / X.400: si on souhaite qu'un message P1 contenant un EDIM soit délivré avant un moment donné, c'est certainement parce que l'EDIM est considéré comme périmé après ce moment. Or, le champ *expiry-time* du ***heading*** de l'EDIM (voir 3.2) est précisément prévu à cette fin. On pourra alors se contenter de spécifier la date et l'heure d'expiration dans ce champ et se passer de cet élément de service. La décision de rejeter un message périmé sera alors prise par l'EDI-UA récepteur et non par le MTS.

43) **Originator Requested Alternate Recipient**

"Elément de service qui permet à l'UA émetteur de spécifier, pour chaque récepteur, un récepteur alternatif à qui le MTS peut délivrer le message si la livraison au récepteur original n'est pas possible."

Cet élément de service est certes utile, mais non indispensable. L'EDI-UA de X.435/ X.400-84 peut le contourner de la façon suivante: l'EDI-UA émetteur envoie le message au récepteur original avec demande de rapport de non-livraison. Si effectivement un tel rapport lui est renvoyé, c'est que la livraison au récepteur original n'est pas possible et l'EDI-UA émetteur peut dès lors envoyer encore une fois le message au récepteur alternatif.

44) **Redirection of Incoming Messages**

"Elément de service qui permet à l'UA de demander au MTS de rediriger les messages qui lui sont adressés vers un autre UA."

Si la redirection de message P1 n'est pas réalisable par le MTS version 1984, on peut certainement la faire à un niveau au-dessus, c'est-à-dire au niveau des EDI-UAs. En effet, on peut demander à l'EDI-UA de relayer la responsabilité de tous les EDIMs en entrée à un autre EDI-UA, avec en plus l'avantage de pouvoir informer l'EDI-UA émetteur du relais du message par le renvoi d'un FN.

45) **Redirection Disallowed by Originator**

"Elément de service qui permet à l'UA émetteur d'informer le MTS de ne pas appliquer la redirection au message soumis si le récepteur a requis la redirection des messages en entrée."

Nous voyons une raison pour laquelle la redirection du message P1 n'est pas souhaitable si le contenu est un EDIM: une telle redirection peut créer des confusions, car une EDIN risque d'être renvoyée par l'EDI-UA d'un récepteur "imprévu" à l'EDI-UA émetteur sans qu'il y ait eu préalablement un renvoi de FN de la part de l'EDI-UA du récepteur original. D'ailleurs, nous trouvons étonnant que CCITT ait prévu la possibilité de la redirection de message P1 dans X.435 (voir l'élément de service 44 ci-dessus).

Pour pallier le manque de cet élément de service dans X.435/X.400-84, nous n'avons

trouvé qu'une solution qui peut paraître étrange, mais qui est la seule possible: si l'EDI-UA émetteur remarque que le message qu'il avait envoyé a été redirigé vers l'EDI-UA d'un récepteur ''imprévu'' (en analysant la séquence des EDINs reçues), et s'il n'est pas satisfait de cette situation (ex.: il avait demandé que le message ne puisse pas être relayé sans acceptation de la responsabilité), alors il peut toujours envoyer un nouvel EDIM à ce récepteur ''imprévu'' pour annuler le premier EDIM. Pour cela, il doit mettre la référence du premier EDIM dans le champ *Obsoleted-EDIMs* du nouvel EDIM, et mettre dans le *primary body* du nouvel EDIM un interchange bidon qui sera rejeté par le récepteur. Mais pour que cela puisse marcher, il faut bien sûr que l'émetteur demande systématiquement toutes les notifications (PN, NN et FN) pour tous les EDIMs qu'il envoie.

C'est une solution ''de bricolage'' et nous en sommes conscients. C'est certainement le seul élément de service dont l'absence pose un vrai problème pratique pour X.435/ X.400-84.

46) **Requested Delivery Method**

''Elément de service qui permet à l'utilisateur du MTS d'indiquer sa préférence de la (des) méthode(s) de livraison de message.''

Cet élément de service est essentiellement utilisé par le PDAU pour indiquer au MTS la manière dont les messages doivent lui être délivrés. Notre discussion dans la catégorie PDAU (voir plus haut) nous permet d'affirmer que cet élément de service est peu utile dans X.435/X.400-84.

47) **Restricted Delivery**

''Elément de service qui permet à l'UA récepteur de demander au MTS de ne pas lui délivrer les messages envoyés par certains UAs émetteurs.''

CCITT précise à l'Annexe B de la Recommandation X.400 (88) que cet élément de service n'est pas encore réalisé dans la forme actuelle de X.400 et qu'il est un sujet d'étude ultérieure. Cela se passe de commentaire.

48) **User/UA Capabilities Registration**

''Elément de service qui permet à l'UA de demander à son MTA de ne pas lui délivrer les messages de certains types de contenu, de certains EITs, ou dont la longueur dépasse une certaine limite.''

Cet élément de service est le seul de cette longue liste qui soit non optionnel: c'est un élément de service de base pour X.435. Son absence paraît donc grave pour X.435/ X.400-84, mais pourtant on ne doit pas trop s'en inquiéter.

En effet, si son absence est théoriquement un problème pour X.435/X.400-84, elle ne l'est pas en pratique. Car c'est un élément de service si essentiel et si utile que la plupart des outils de développement de X.400-84 l'ont implémenté d'une manière ou d'une autre. C'est notamment le cas du *X.400 Gateway API* développé par la société

Retix et utilisé pour l'implémentation de notre prototype X.435/X.400-84. Nous en parlerons plus longuement dans le chapitre 5.

4.6 Résumé de l'analyse

Nous avons ainsi analysé les différents éléments qui risquent de poser un problème à une application X.435/X.400-84. Résumons à présent le résultat de notre analyse.

Le paragraphe 4.1 nous informe que le *Message Store* peut rendre la vie d'un utilisateur EDI plus facile en fournissant des facilités telles que la recherche sélective et l'*auto-action*. Mais ce concept est seulement optionnel dans X.435. Bien que son absence soit regrettable, cela n'est pas un obstacle réel pour X.435/X.400-84.

Le paragraphe 4.2 nous informe que les services de répertoire X.500 peuvent être utilisés par X.435/X.400-84, même si cela implique l'implémentation ou l'acquisition des éléments de service ACSE et ROSE.

Mieux encore, le paragraphe 4.4 nous informe que **CCITT reconnaît implicitement l'utilisation de X.400-84 pour X.435** en nous montrant les actions à prendre pour une telle réalisation. Cela est plus qu'encourageant, d'autant que le paragraphe 4.5 nous informe que la quasi totalité des éléments de service manquants de X.400-84 - excepté l'élément de service "Redirection Disallowed by Originator" - peuvent facilement être compensés d'une manière ou d'une autre.

Malheureusement, notre discussion dans le paragraphe 4.3 nous montre que le problème provient du manque des services de sécurité dans X.400-84. Cela ne semble pas catastrophique à première vue, puisque les services de sécurité de X.435 sont optionnels et l'utilisateur EDI peut librement choisir de ne pas les utiliser. Mais cela a des implications profondes comme l'a montré 4.3.4.

En résumé, nous pouvons affirmer que **l'utilisation de X.400-84 pour X.435 est théoriquement faisable, même si l'absence des services de sécurité le rend très irréaliste en pratique.**

Chapitre 5

Implémentation du Prototype X.435/X.400-84

Dans le chapitre précédent, nous avons vu que l'implémentation d'une application de X.435 basée sur X.400 version 1984 est faisable, malgré les quelques inconvénients mis en évidence.

Une implémentation d'un prototype X.435/X.400-84 a ainsi été tentée pour mettre en pratique notre analyse. Nous avons pris la décision d'ignorer l'absence des services de sécurité dans X.400-84 et de ne pas prendre en compte pour le moment l'éventuelle intégration du prototype avec X.500. Concernant l'absence du Message Store, le problème sera partiellement contourné par l'installation d'un serveur central pour éviter tout débordement de messages. Nous allons présenter dans ce chapitre les différents outils et APIs utilisés, la mise en oeuvre de ce prototype, ainsi que les problèmes rencontrés au cours de l'implémentation.

5.1 Introduction à la notion d'API

Nous utilisons deux APIs pour le développement du prototype X.435/X.400-84. Ainsi, il nous paraît intéressant d'introduire d'abord quelques notions sur les APIs.

Un API (*Application Program Interface*) est, comme son nom l'indique, une interface qui permet à un programme d'accéder à certains services. L'interface comporte un ensemble de points d'accès à travers lesquels un programme peut accéder aux services de niveau inférieur sans devoir se préoccuper du traitement interne de ceux-ci.

Conceptuellement, un API sépare deux univers différents. Chaque univers ignore le fonctionnement et la structure interne de l'autre, mais les deux arrivent à se comprendre et à communiquer via l'API. Pour prendre un exemple plus concret, un API de X.400 (dont nous allons discuter en détail) se trouve entre deux univers de traitement de message, c'est-à-dire entre celui de X.400 et celui du système de messagerie local. Les deux systèmes arrivent à se communiquer et à se passer des messages grâce aux actions de traduction de l'API. Ces actions peuvent être la traduction de représentation, la traduction d'adresse, ... etc.

Matériellement, un API consiste en des librairies de fonctions que les programmeurs utilisent pour bâtir leurs programmes d'application. On rencontre les APIs surtout dans les domaines de système d'exploitation et de télécommunication, où une division de tâches en plusieurs couches est nette. Nous en utilisons deux pour

l'implémentation du prototype : la version 84 du *X.400 Gateway API* de la société RETIX (Santa Monica, USA), et le *X.435 API* que nous avons spécialement créé pour le besoin du prototype.

5.2 *X.400 Gateway API*

CCITT définit dans ses recommandations les règles de traitement à suivre et la structure des PDUs à respecter. Les recommandations consistent en des ouvrages comprenant des centaines (voire des milliers) de pages rédigées en anglais. Le risque est grand que les développeurs de programmes basés sur les Recommandations CCITT interprètent ces dernières chacun à sa manière et écrivent des applications incompatibles entre elles. D'où la nécessité pour les vendeurs d'ordinateurs, de logiciels et de systèmes de communication de collaborer pour harmoniser leurs approches aux Recommandations CCITT.

En décembre 1988, une vingtaine de sociétés de logiciels et de systèmes de télécommunication ont créé la *X.400 API Association*. Cette association a pour but de réunir les efforts des différentes sociétés participantes afin de définir une interface commune dans le développement de programmes X.400. Les travaux ont abouti en juin 1989 pour la version 1984 de X.400 sous forme d'un document nommé *X.400 Gateway API Specification*. C'est donc en quelque sorte une ''Recommandation sur la Recommandation X.400'' qui régit l'approche pratique à X.400, et que tous ceux qui voudraient développer des applications X.400 devront respecter.

Certains lecteurs peuvent réagir en apprenant que des ''Recommandations sur des Recommandations'' sont nécessaires pour développer des applications du modèle OSI. Cela est simplement dû à la grande complexité de celles-ci. En effet, cette complexité constitue en même temps un avantage et un inconvénient. Des applications telles que X.400 ou FTAM sont très puissantes et offrent infiniment plus de possibilités que leurs homologues du modèle TCP/IP; mais elles sont si complexes et fournissent tant de possibilités optionnelles que leurs développements et mises au point sont rendus ardus et très difficiles. Cela n'est certainement pas étranger au retard de la généralisation de l'OSI et constitue le tendon d'Achille de celui-ci. Pour preuve, le peu d'applications OSI existantes aujourd'hui ont souvent du mal à se communiquer entre elles, alors que les applications TCP, bien plus simples, fonctionnent à merveille. Espérons que grâce à ces spécifications d'API, les applications OSI se stabiliseront d'ici quelques années.

5.2.1 Définition du *X.400 Gateway API*

La spécification de *X.400 Gateway API* spécifie une interface programmatique qui divise la fonctionnalité d'un X.400 MTA en deux composants de logiciel distincts, *Mail System Gateway* et *X.400 Gateway Service*. Communément, *Mail System Gateway*, *X.400 Gateway Service* et *X.400 Gateway API* sont respectivement appelés *client*, *service* et *interface*. [X.400-API] (Voir figure 5-1)

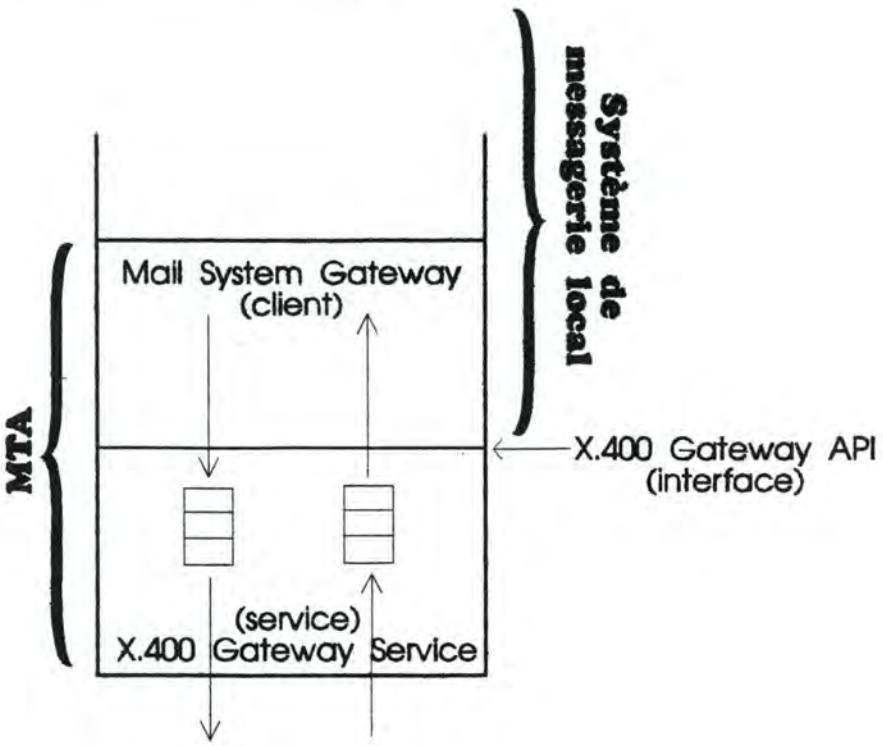

Figure 5-1 : Modèle fonctionnel du X.400 Gateway API

Le *client*, c'est-à-dire la partie du MTA faisant partie d'un système de messagerie local, accède au *service* à travers *l'interface* pour envoyer et recevoir des messages, des rapports de livraison ou des *probes*. Concrètement, le *client* utilise les fonctions fournies par *l'interface* pour confier/recevoir des messages, des rapports de livraison ou des *probes* au/du *service*.

Le *service* a pour rôle de se charger du problème de routage pour relayer les messages que lui a confiés le *client* au MTA destinataire, et du problème d'encodage/ décodage ASN.1 des messages. Il travaille de façon transparente pour le *client* et épargne ainsi à ce dernier ces travaux fastidieux.

L'interface a deux rôles principaux:

1) Elle définit les différentes classes d'objets qui constituent les messages P1 et P2. De par la position de l' *interface* (à l'intérieur du MTA), le mélange fait ici entre la notion de message P2 et le MTA peut paraître surprenant à première vue, mais la raison en est simple: l' *interface* est chargée de passer au *service* des objets représentant un message P1 et contenant des objets représentant à leur tour un message P2. Pour cela, elle doit nécessairement connaître la structure des classes d'objet qui représentent un message P2. La composition d'un message P2 reste bien entendu le travail de l'UA.

2) Elle fournit les fonctions de manipulation d'objets pour le *client*. Ces fonctions permettent par exemple de créer un objet, d'insérer un objet dans un autre, ou de transférer un objet vers son destinataire via le *service*. (La relation entre ces fonctions et les services abstraits de X.400-84 tels que *submit* ou

deliver sera expliquée dans 5.2.3)

Nous allons maintenant discuter en détail le fonctionnement de l' *interface.*

5.2.2 Classes d'objet du *X.400 Gateway API*

La spécification de l' *interface* définit les messages P1 et P2 selon une approche orientée-objet. La notion de classe et d'objet est utilisée pour la manipulation des messages. Un objet, c'est-à-dire une instance d'une classe, regroupe des attributs et peut à son tour contenir d'autres objets. Cela signifie qu'un message P1 ou P2 correspond à un objet qui comporte d'autres objets, et ainsi de suite. Un message est donc une structure d'objets imbriqués.

Par exemple, nous avons ''O/R Name'' qui est une classe d'objet qui comporte 14 attributs, tous de type ''PrintableString'', et ''Delivery Envelope'' qui est une classe d'objet qui comporte 12 attributs, dont 4 sont de type ''O/R Name''. Pour qu'on ait une idée plus précise, nous donnons ici la description complète de la classe d'objet ''Recipient Descriptor'' telle qu'elle se trouve dans [X.400-API]:

Attribute	Value syntax	Value length	Value number	Value initially
Explicit Conversion	Enum(Explicit Conversion)	-	1	no-conversion
MTA Report Request	Enum(Report Request)	-	1	non-delivery
MTA Responsibility	Boolean	-	1	true
Originator Report Request	Enum(Report Request)	-	1	non-delivery
Recipient Name	Object(O/R Name)	-	1	-
Recipient Number	Integer	-	1	-

Attribute = Le nom de l'attribut.

Value syntax = Le type de l'attribut. Cela peut être un type prédéfini dans ASN.1, ou le type ''Object'' qui signifie que l'attribut est un objet.

Value length = La longueur de l'attribut. Significatif uniquement pour les attributs de type ''String''.

Value number = Le nombre de répétition de l'attribut dans l'objet.

Value initially = La valeur par défaut de l'attribut.

Les lecteurs attentifs remarqueront que la classe d'objet ''Recipient Descriptor'' correspond en réalité à la définition en ASN.1 du type ''RecipientInfo'' d'un PDU P1 (page 151 du [X.400-84]). La *X.400 API Association* a simplifié cette dernière en enlevant quelques niveaux d'imbrication pour en faire une classe d'objet plus compréhensible aux programmeurs. Il y a cependant quelques changements de nom d'attribut par rapport à la définition initiale, mais cela est resté sans explication de la part de la *X.400 API Association.*

Il y a deux catégories de classes d'objets. La catégorie MT (Message Transfer) concerne les classes d'objet relatives au message P1, et la catégorie IM (Interpersonal Messaging) concerne celles relatives au message P2. Néanmoins, certaines classes

d'objet apparaissent dans les deux catégories; c'est notamment le cas de la classe "O/R Name". On peut bien sûr trouver dans [X.400-API] la liste et la description complète des différentes classes d'objet de *X.400 Gateway API*.

5.2.3 Fonction du *X.400 Gateway API*

L'*interface* fournit des fonctions pour manipuler les objets. Ces fonctions permettent au *client* et au *service* de se communiquer et jouent le rôle du traducteur. Les traductions effectuées par l'*interface* sont par exemple:

-Traduction d'adresse (Adresse locale pour le *client*, *O/R Name* pour le *service*)

-Traduction de représentation (Objet pour le *client*, ASN.1 pour le *service*)

Il y a deux catégories de fonctions: la catégorie OM (Object Management) concerne les fonctions de création, d'effacement et d'examen d'objet, et la catégorie MT (Message Transfer) concerne les fonctions d'envoi et de réception d'objets.

La catégorie OM comporte 14 fonctions. Voici les plus importantes parmi elles: [RETIX]

om_create() : Créer un objet.

om_insert() : Insérer un attribut dans un objet.

om_remove() : Retirer un attribut d'un objet.

om_replace() : Remplacer la valeur d'un attribut par une autre dans un objet.

om_fetch() : Consulter la valeur d'un attribut dans un objet.

om_check() : Vérifier qu'un objet satisfait syntaxiquement la spécification de sa classe.

om_copy() : Dupliquer un objet.

om_delete() : Effacer un objet.

La catégorie MT comporte 8 fonctions que voici: [RETIX]

mt_open(): Etablir une session de travail entre le *client* et le *service*.

mt_close(): Terminer une session de travail entre le *client* et le *service*.

mt_transfer_out() : Envoyer un objet (correspondant à un message P1) composé par le *client* vers l'extérieur. L'objet est mis dans la file de sortie gérée par le *service*.

mt_wait() : Attendre l'indication qu'un objet soit arrivé dans la file d'entrée gérée par le *service.*

mt_start_transfer_in() : Recevoir un objet (correspondant à un message P1, c'est-à-dire un message, un rapport ou un *probe*) de la file d'entrée. Cette fonction "réserve" l'objet et retourne un *handler* (pointeur en langage C) au *client.*

mt_finish_transfer_in() : Effacer un objet réservé de la file d'entrée après que celui-ci ait été traité par le *client.*

mt_cancel_transfer_in() : Invalider l'effet de la fonction mt_start_transfer_in() en annulant la réservation d'un ou de tous les objets réservés de la file d'entrée.

mt_size() : Examiner la file d'entrée pour déterminer les nombres d'objets réservés et non-réservés.

Il est intéressant de comparer ces fonctions MT avec les services abstraits de la Recommandation X.411 version 84. Les services abstraits sont des services fournis par le MTA à l'UA pour l'envoi et la réception des messages, ainsi que pour le contrôle du MTA par l'UA. Nous allons donner la liste complète de ces services abstraits, accompagnés d'explications sur la manière dont ils sont (éventuellement) réalisés par les fonctions MT du *X.400 Gateway API* de Retix:

Logon : Réalisé par mt_open().

Logoff : Réalisé par mt_close().

Register : Réalisé uniquement par les outils de configuration fournis par Retix.

Control : Non réalisable. On est obligé d'utiliser *Register.*

Submit : Réalisé par mt_transfer_out().

Probe : Réalisé par mt_transfer_out().

Deliver : Réalisé par mt_start_transfer_in() puis mt_finish_transfer_in().

Notify : Réalisé par mt_start_transfer_in() puis mt_finish_transfer_in().

Cancel : Réalisé par mt_cancel_transfer_in() uniquement pour l'annulation de la livraison d'un message. L'annulation de la soumission d'un message n'est pas réalisable.

Change-password : Réalisé uniquement par les outils de configuration fournis par Retix.

En résumé, voici comment fonctionne le *X.400 Gateway API*: pour envoyer un message interpersonnel, il suffit de créer un objet IM (message P2) comportant ses sous-objets à l'aide des fonctions OM, de créer ensuite un objet MT (message P1) qui inclut l'objet IM toujours à l'aide des fonctions OM, puis de transférer l'objet MT à son destinataire à l'aide des fonctions MT. La réception d'un message interpersonnel suit un raisonnement similaire.

La raison pour laquelle X.400 présente uniquement la possibilité d'envoyer des messages interpersonnels est purement historique, car on n'avait pas envisagé d'autres usages avant l'avènement de X.435. Dans le cas de notre prototype, nous n'avons évidemment pas besoin des classes d'objet IM, puisque il s'agit d'envoyer des EDIMs (Pedi) et non des messages interpersonnels (P2). Il nous est donc nécessaire de créer une spécification de classes d'objet pour X.435.

5.3 X.435 API

La Recommandation X.435 étant assez récente, les APIs de X.400 disponibles actuellement n'ont pas encore prévu la possibilité de manipuler les messages Pedi et ne gèrent que les messages P2. Il semblerait cependant qu'une version de *X.435 API* soit déjà disponible sur le marché à la fin de l'année 1991, mais vu que la Recommandation X.435 disponible au moment de ce texte n'est qu'une "Draft Recommendation" - donc une version provisoire, on peut douter de l'intérêt de cet API qui est susceptible de changements futurs et qui, soit dit en passant, est très onéreux.

Pour cette raison, l'auteur de ce texte a créé son propre *X.435 API*, basé sur les mêmes principes de *X.400 Gateway API Specification*. Le but de ce *X.435 API* est de:

1) Spécifier les différentes classes d'objet pour la construction d'un message Pedi.

2) Fournir les fonctions qui permettent la création et la manipulation des objets EDI pour ensuite les transférer à leur destinataire via *X.400 Gateway API*.

Nous allons voir quelles sont les différentes étapes du développement de notre *X.435 API*.

5.3.1 Définition des classes d'objet EDI

Pour définir les différentes classes d'objet EDI à la manière du *X.400 Gateway API*, il faut d'abord analyser le PDU Pedi (X.435). Ce dernier est d'une très grande complexité, comportant parfois une dizaine de niveaux d'imbrication et tient sur une quinzaine de pages en langage ASN.1.

L'analyse du PDU Pedi n'est pas simple. Tout d'abord, la définition en ASN.1 du

PDU Pedi telle qu'elle est présentée dans [X.435] comporte des erreurs d'inattention et de références introuvables (On trouve la liste de ces erreurs à l'Annexe A). Un travail de "débroussaillement" a donc été nécessaire pour obtenir une version correcte du PDU Pedi (On trouve cette version corrigée du PDU Pedi à l'Annexe B).

Ensuite, les nombreux niveaux d'imbrication de la définition en ASN.1 du PDU Pedi rendent la compréhension de ce dernier quasi impossible. Le langage ASN.1 emploie des références par l'identifiant d'une définition de type. On saute ainsi d'une définition à une autre; dans le cas du PDU Pedi, le parcours semble interminable. En partant de la racine, on "s'enfonce" de plus en plus dans les différents niveaux de définitions; on perd très vite le fil de pensée et on finit par se demander où on se trouve. Alors que si on avait une définition du PDU Pedi où on met en évidence la structure générale de l'ensemble et où on supprime les sauts de référence, une compréhension de la situation globale serait beaucoup plus aisée (Voir figure 5-2). Un autre travail a donc été réalisé pour transformer la définition initiale du PDU Pedi en une autre structure **beaucoup plus compréhensible par l'analyste** (On trouve le résultat de cette transformation à l'Annexe C).

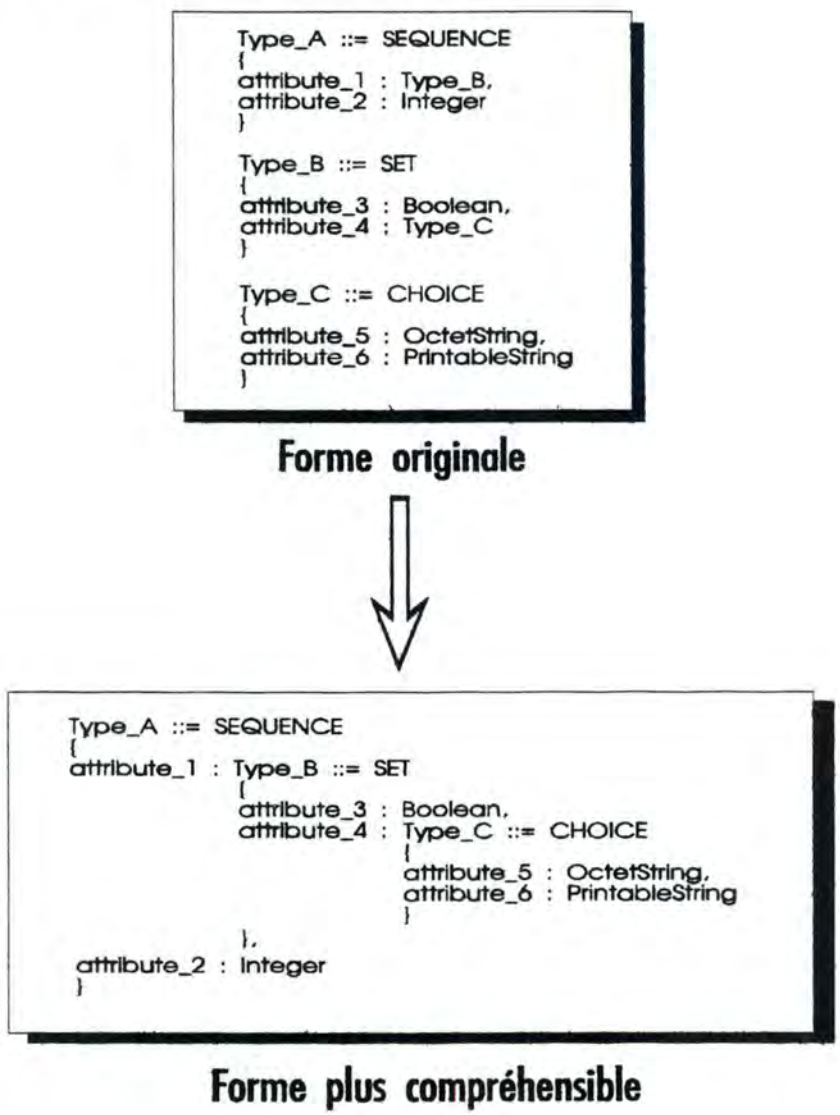

Figure 5-2 : Transformation de la structure du langage ASN.1

Après ces deux travaux de préparation, nous sommes enfin prêts à entamer le travail sérieux, c'est-à-dire à dégager les différentes classes d'objet EDI. Mais comment y parvenir en ayant la version corrigée du PDU Pedi en mains et en ayant maintenant une meilleure vue globale sur le PDU Pedi?

Dans la *X.400 Gateway API Specification* discutée précédemment, on nous a simplement présenté les classes d'objet sans nous expliquer selon quels critères celles-ci ont été obtenues. Bien sûr, on voit intuitivement qu'une classe d'objet correspond grosso modo à une définition de type SEQUENCE, SET ou CHOICE du PDU P1 et P2, mais on voit également des cas où plusieurs définitions de type sont regroupées dans une seule classe d'objet. En outre, les noms d'attribut des définitions de type du PDU sont souvent changés dans les classes d'objet sans raisons apparentes. On a l'impression que la *X.400 API Association* a voulu mettre l'accent autant sur la sémantique que sur la syntaxe des définitions de type du PDU pour dégager les classes d'objet, mais il est regrettable qu'elle n'apporte pas plus de précisions sur la manière dont elle a élaboré la *X.400 Gateway API Specification*.

Après ces considérations, voici comment nous avons procédé à la définition des classes d'objet du *X.435 API*:

1) Chaque définition de type SEQUENCE, SET ou CHOICE du PDU Pedi est considérée comme une classe d'objet EDI.

2) Les identifiants des définitions de type et les noms des attributs du PDU Pedi restent inchangés dans la spécification des classes d'objet EDI.

3) Le nombre de répétitions (''Value number'') d'un attribut dépend de son type dans le PDU Pedi. Il est normalement de 1, de 0 à 1 si son type est suivi du mot-clé OPTIONAL, indéterminé s'il est de type SEQUENCE OF ou SET OF.

4) La longueur d'un string (''Value length'') est déterminée à partir du mot-clé SIZE(...). De même, la valeur par défaut (''Value initially'') est déterminée à partir du mot-clé DEFAULT {...}.

Cette façon de procéder peut sembler un peu simpliste, mais elle est voulue. Car l'auteur de ce texte n'a pas la prétention de vouloir se substituer à la vingtaine de sociétés qui forment la *X.400 API Association* pour la spécification du *X.435 API*. Notre modeste but est pour le moment de trouver une interface de programmation acceptable pour le développement d'une application X.435, en attendant qu'une version définitive de *X.435 API Specification* soit disponible au public.

Voici donc la liste complète des classes d'objet de notre *X.435 API*:

Attribute	Value syntax	Value length	Value number	Value initially
CLASS InformationObject				
edim	Object(EDIM)	-	0-1	-
edin	Object(EDIN)	-	0-1	-
CLASS EDIM				
heading	Object(Heading)	-	1	-
body	Object(Body)	-	1	-
CLASS Heading				
this_EDIM	Object(EDIMIdentifier)	-	1	-
originator	Object(ORName)	-	0-1	-
recipients	Object(Recipients)	-	0 or more	-
edin_receiver	Object(EDINReceiver)	-	0-1	-
responsability_forwarded	Boolean	-	1	FALSE
edi_bodypart_type	String(ObjectIdentifier)	0-32	1	''2.6.7.11.0''
incomplete_copy	Boolean	-	1	FALSE
expiry_time	String(Time)	0-17	0-1	-
related_messages	Object(RelatedMessages)	-	0 or more	-
obsoleted_EDIMs	Object(EDIMIdentifier)	-	0 or more	-
edi_application_security_elements	Object(EDIApplicationSecurityElements)	-	0-1	-
cross_referencing_information	Object(CrossReferencingInformation)	-	0 or more	-
edi_message_type	String(T61)	1-6	0 or more	-
service_string_advice	Object(ServiceStringAdvice)	-	0-1	-
syntax_identifier	Object(SyntaxIdentifier)	-	0-1	-
interchange_sender	Object(InterchangeSender)	-	0-1	-
date_and_time_of_preparation	String(Time)	0-17	0-1	-
application_reference	String(T61)	1-14	0-1	-
CLASS EDIMIdentifier				
user	Object(ORName)	-	1	-
user_relative_identifier	String(Printable)	0-64	1	-
CLASS ORName				
country_name	String(Printable)	2 or 3	0-1	(local)
ADMD_name	String(Printable)	0-16	0-1	(local)
network_address	String(Printable)	1-15	0-1	-
terminal_identifier	String(Printable)	1-24	0-1	-
PRMD_name	String(Printable)	1-16	0-1	-
organization_name	String(Printable)	1-64	0-1	-
numeric_user_identifier	String(Printable)	1-32	0-1	-
surname	String(Printable)	1-40	0-1	-
given_name	String(Printable)	1-16	0-1	-
initials	String(Printable)	1-5	0-1	-
generation_qualifier	String(Printable)	1-3	0-1	-
organizational_unit_name	String(Printable)	1-32	0-4	-
domain_defined_attributes	Object(DomainDefinedAttributes)	-	0-4	-
directory_name	Object(DirectoryName)	-	0 or more	-
CLASS DomainDefinedAttributes				
domain_type	String(Printable)	1-8	1	-
domain_value	String(Printable)	1-128	1	-
CLASS DirectoryName				
attribute_type	String(ObjectIdentifier)	0-32	1	-
attribute_value	Any	-	1	-
CLASS Recipients				
recipient	Object(ORName)	-	1	-
action_request	String(ObjectIdentifier)	0-32	1	''2.6.7.13.0''
edi_notification_requests	Object(EDINotificationRequests)	-	0-1	-
responsability_passing_allowed	Boolean	-	1	FALSE
interchange_recipient	Object(InterchangeRecipient)	-	0-1	-
recipient_reference	Object(RecipientReference)	-	0-1	-
interchange_control_reference	String(T61)	1-14	0-1	-
processing_priority_code	String(T61)	1	0-1	-
acknowledgement_request	Boolean	-	1	FALSE
communications_agreement_id	String(T61)	1-35	0-1	-
test_indicator	Boolean	-	1	FALSE
authorization_information	Object(AuthorizationInformation)	-	0-1	-
CLASS EDINotificationRequests				
edi_notification_requests	String(Bit)	0-16	1	{}
edi_notification_security	String(Bit)	0-16	1	{}
edi_reception_security	String(Bit)	0-16	1	{}

CLASS InterchangeRecipient

recipient_identification_code	String(T61)	1-35	1	-
identification_code_qualifier	String(T61)	1-4	0-1	-
routing_address	String(T61)	1-14	0-1	-

CLASS RecipientReference

recipient_reference	String(T61)	1-14	1	-
recipient_reference_qualifier	String(T61)	1-2	0-1	-

CLASS AuthorizationInformation

authorization_information	String(T61)	1-10	1	-
authorization_information_qualifier	String(T61)	1-2	0-1	-

CLASS EDINReceiver

edin_receiver_name	Object(ORName)	-	1	-
original_edim_identifier	Object(EDIMIdentifier)	-	0-1	-
first_recipient	Object(ORName)	-	0-1	-

CLASS RelatedMessages

edi_message_reference	Object(EDIMIdentifier)	-	0-1	-
external_message_reference	External	-	0-1	-

CLASS EDIApplicationSecurityElements

edi_application_security_element	String(Bit)	0-8191	0-1	-
edi_encrypted_primary_bodypart	Boolean	-	0-1	-

CLASS CrossReferencingInformation

application_cross_reference	String(Octet)	-	1	-
message_reference	Object(EDIMIdentifier)	-	0-1	-
body_part_reference	Integer	-	1	-

CLASS ServiceStringAdvice

component_data_element_separator	String(Octet)	1	1	-
data_element_separator	String(Octet)	1	1	-
decimal_notation	String(Octet)	1	1	-
release_indicator	String(Octet)	1	0-1	-
reserved	String(Octet)	1	0-1	-
segment_terminator	String(Octet)	1	1	-

CLASS SyntaxIdentifier

syntax_identifier	String(T61)	1-4	1	-
syntax_version	String(T61)	1-5	1	-

CLASS InterchangeSender

sender_identification	String(T61)	1-35	1	-
identification_code_qualifier	String(T61)	1-4	0-1	-
address_for_reverse_routing	String(T61)	1-14	0-1	-

CLASS Body

primary_body_part	Object(PrimaryBodyPart)	-	1	-
additional_body_parts	Object(EDIMExternallyDefinedBodyPart)	-	0 or more	-

CLASS EDIMExternallyDefinedBodyPart

body_part_reference	Integer	-	0-1	-
parameters	External	-	0-1	-
data	External	-	1	-

CLASS PrimaryBodyPart

edi_body_part	String(Octet)	-	0-1	-
forwarded_EDIM	Object(EDIMBodyPart)	-	0-1	-

CLASS EDIMBodyPart

parameters	Object(MessageParameters)	-	0-1	-
data	Object(MessageData)	-	1	-

CLASS MessageParameters

delivery_time	String(Time)	0-17	0-1	-
delivery_envelope	Object(OtherMessageDelivery)	-	0-1	-
other_parameters	String(IA5)	1-256	0-1	-

CLASS OtherMessageDelivery

content_type	Object(DeliveredContentType)	-	1	-
originator_name	Object(ORName)	-	1	-
original_EITs	Object(EITs)	-	0-1	-
priority	Enum(Priority)	-	1	normal
delivery_flags	String(Bit)	0-16	0-1	-
other_recipient_names	Object(ORName)	-	0-32767	-
this_recipient_name	Object(ORName)	-	1	-
originally_intended_recipient_name	Object(ORName)	-	0-1	-
converted_EITs	Object(EITs)	-	0-1	-
message_submission_time	String(Time)	0-17	1	-
content_identifier	String(Printable)	1-16	0-1	-

CLASS DeliveredContentType

built_in	Integer	0-32767	0-1	-
external	String(ObjectIdentifier)	0-32	0-1	-

CLASS EITs

built_in_EITs	String(Bit)	0-32	1	-
g3_fax	String(Bit)	-	1	{}
teletex	Object(TeletexNonBasicParameters)	-	1	{}
g4_class_1_and_mixed_mode	Any	-	0-1	-
external_EITs	String(ObjectIdentifier)	0-32	0-1024	-

CLASS TeletexNonBasicParameters

graphic_character_sets	String(Teletex)	-	0-1	-
control_character_sets	String(Teletex)	-	0-1	-
page_formats	String(Octet)	-	0-1	-
miscellaneous_terminal_capabilities	String(Teletex)	-	0-1	-
private_use	String(Octet)	0-128	0-1	-

CLASS MessageData

heading	Object(Heading)	-	1	-
body	Object(BodyOrRemoved)	-	1	-

CLASS BodyOrRemoved

primary_or_removed	Object(PrimaryOrRemoved)	-	1	-
additional_body_parts	Object(AdditionalBodyParts)	-	0 or more	-

CLASS PrimaryOrRemoved

removed_edi_body	Null	-	0-1	-
primary_body_part	Object(PrimaryBodyPart)	-	0-1	-

CLASS AdditionalBodyParts

external_body_part	Object(EDIMExternallyDefinedBodyPart)	-	0-1	-
place_holder	Object(EDIMExternallyDefinedBodyPart)	-	0-1	-

CLASS EDIN

positive_notification	Object(PositiveNotification)	-	0-1	-
negative_notification	Object(NegativeNotification)	-	0-1	-
forwarded_notification	Object(ForwardedNotification)	-	0-1	-

CLASS PositiveNotification

pn_common_fields	Object(CommonFields)	-	1	-
pn_supplementary_information	String(IA5)	1-256	0-1	-

CLASS NegativeNotification

nn_common_fields	Object(CommonFields)	-	1	-
nn_reason_code	Object(NNReasonCode)	-	1	-
nn_supplementary_information	String(IA5)	1-256	0-1	-

CLASS NNReasonCode

nn_ua_ms_reason_code	Object(NNUAMSReasonCode)	-	0-1	-
nn_user_reason_code	Object(NNUserReasonCode)	-	0-1	-
nn_pdau_reason_code	Object(NNPDAUReasonCode)	-	0-1	-

CLASS NNUAMSReasonCode

nn_ua_ms_basic_code	Integer	0-32767	1	-
nn_ua_ms_diagnostic	Integer	1-32767	0-1	-

CLASS NNUserReasonCode

nn_user_basic_code	Integer	0-32767	1	-
nn_user_diagnostic	Integer	-	0-1	-
CLASS NNPDAUReasonCode				
nn_pdau_basic_code	Integer	0-32767	1	-
nn_pdau_diagnostic	Integer	1-32767	0-1	-
CLASS ForwardedNotification				
fn_common_fields	Object(CommonFields)	-	1	-
forwarded_to	Object(ORName)	-	1	-
fn_reason_code	Object(FNReasonCode)	-	1	-
fn_supplementary_information	String(IA5)	1-256	0-1	-
CLASS FNReasonCode				
fn_ua_ms_reason_code	Object(FNUAMSReasonCode)	-	0-1	-
fn_user_reason_code	Object(FNUserReasonCode)	-	0-1	-
fn_pdau_reason_code	Object(FNPDAUReasonCode)	-	0-1	-
CLASS FNUAMSReasonCode				
fn_ua_ms_basic_code	Integer	0-32767	1	-
fn_ua_ms_diagnostic	Integer	0-32767	0-1	-
fn_security_check	Boolean	-	1	FALSE
CLASS FNUserReasonCode				
fn_user_basic_code	Integer	0-32767	1	-
fn_user_diagnostic	Integer	-	0-1	-
CLASS FNPDAUReasonCode				
fn_pdau_basic_code	Integer	0-32767	1	-
fn_pdau_diagnostic	Integer	-	0-1	-
CLASS CommonFields				
subject_edim	Object(EDIMIdentifier)	-	1	-
edin_originator	Object(ORName)	-	1	-
first_recipient	Object(ORName)	-	0-1	-
notification_time	String(Time)	0-17	1	-
notification_security_elements	Object(SecurityElements)	-	0-1	-
edin_initiator	Enum(EDINInitiator)	-	1	-
CLASS SecurityElements				
original_content	String(Octet)	-	0-1	-
edi_application_security_elements	Object(EDIApplicationSecurityElements)	-	0-1	-

Un objet de la classe ''InformationObject'' correspond donc à un PDU Pedi. Pour obtenir un tel objet, il faut adopter une approche ''bottom-up'': on crée d'abord des objets du dernier niveau, puis on les insère dans un objet du niveau supérieur, et ainsi de suite. Pour prendre une vue plus imagée, c'est avec des briques de base qu'on construit petit à petit la grande maison qui correspond à un objet ''InformationObject''. Après la composition d'un tel objet, on peut alors le confier au *X.400 Gateway API* qui l'inclut dans un objet MT (message P1) et transfère le tout vers son destinataire à l'aide des fonctions MT.

5.3.2 Fonctions du *X.435 API*

Toujours à la manière de *X.400 Gateway API*, notre *X.435 API* fournit des fonctions de manipulation d'objets. Le nombre de ces fonctions est ici restreint, il s'agit du strict nécessaire pour manipuler un objet EDI. *X.400 Gateway API* fournit bien d'autres fonctions supplémentaires (Voir 5.2.3) qui sont certes utiles, mais non indispensables. Nous ne les avons pas reprises dans notre *X.435 API*.

Les fonctions EDI du *X.435 API* sont:

k_create() : Créer un objet EDI.

k_insert() : Insérer un attribut dans un objet EDI.

k_fetch() : Consulter la valeur d'un attribut dans un objet EDI.

k_check() : Vérifier si un objet EDI satisfait syntaxiquement la spécification de sa classe.

k_copy() : Dupliquer un objet EDI.

k_delete() : Effacer un objet EDI.

La figure 5-3 montre la relation entre *X.400 Gateway API* et *X.435 API*. Le rôle de *X.435 API* est de permettre aux programmeurs de construire des objets EDI grâce à la spécification des classes d'objet EDI et les fonctions EDI listées ci-dessus. Ensuite, l'objet EDI qui correspond à un message Pedi (l'objet "InformationObject") est

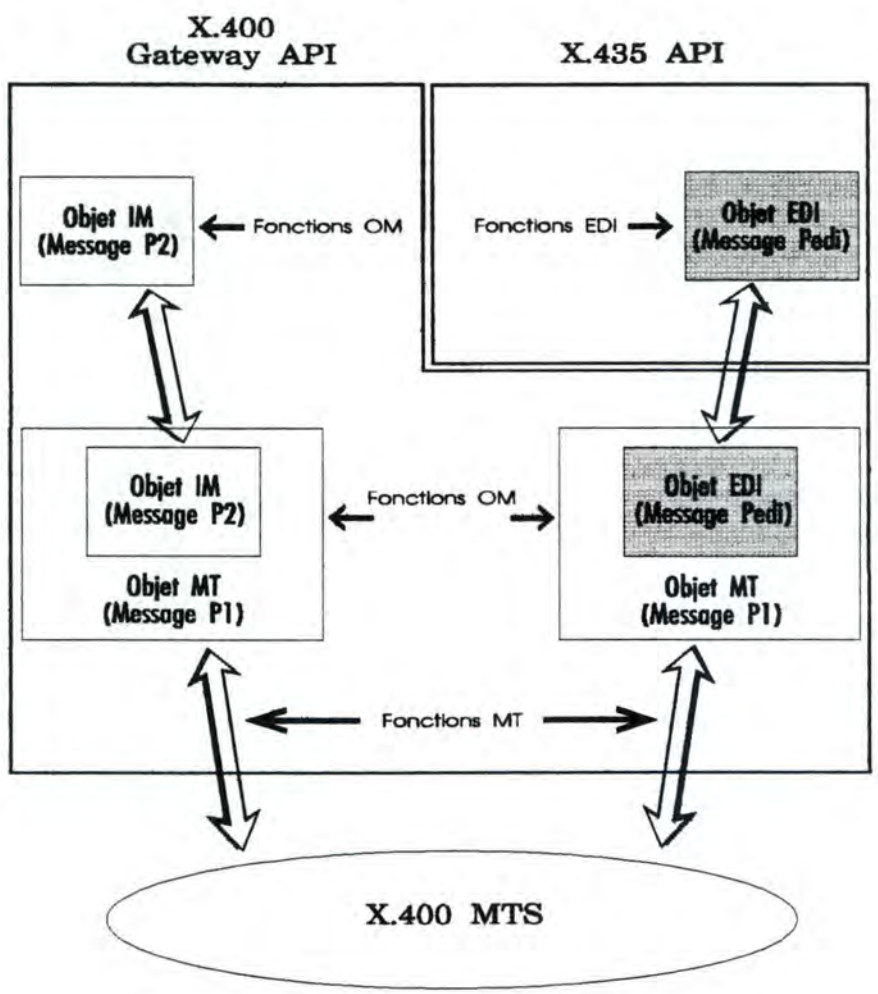

Figure 5-3 : Relation entre X.400 Gateway API et X.435 API

confié à *X.400 Gateway API* pour le transfert vers le récepteur.

5.3.3 Du langage ASN.1 à l'encodage BER

La *X.400 Gateway API Specification* définit uniquement les classes d'objet et les paramètres et valeurs de retour des fonctions OM et MT. La structure interne des objets peut être choisie librement par les développeurs des *X.400 Gateway APIs*. Le *X.400 Gateway API* de Retix que nous utilisons a une structure interne extrêmement complexe et difficile à déchiffrer. Pour cette raison, nous avons décidé d'utiliser une autre structure interne beaucoup plus simple pour les objets EDI de notre *X.435 API*.

Par conséquent, on ne peut donner un objet EDI de notre *X.435 API* tel quel au *X.400 Gateway API*, car les deux *APIs* n'utilisent pas la même structure interne et le *X.400 Gateway API* est incapable de reconnaître un objet EDI. Il faut préalablement encoder l'objet EDI suivant les règles d'encodage du langage ASN.1 (voir X.409 (X.400-84) ou X.208 & X.209 (X.400-88) pour ASN.1) avant de passer le résultat de l'encodage BER présenté comme une suite d'octets au *X.400 Gateway API* pour la transmission au destinataire.

Le langage ASN.1 utilisé dans le modèle OSI est, comme son nom l'indique, une notation de syntaxe abstraite. Pour passer à une forme concrète, il faut suivre des règles de traitement afin de convertir une structure ASN.1 correctement garnie en encodage BER. Le problème n'est pas simple vu la complexité du langage ASN.1, et le développement d'un outil de conversion constitue à lui seul un travail aussi complexe que le développement du prototype X.435/X.400-84.

Heureusement, de tels outils existent déjà sur le marché. Le plus utilisé parmi eux est certainement le compilateur *pepsy* de l'ISODE.

ISODE [ISODE]

ISODE signifie *ISO Development Environment*. C'est un ensemble de logiciels développés comme outils de recherche et qui représentent un effort pour promouvoir l'utilisation de l'OSI. Les trois premières versions de l'ISODE ont été développées au Northrop Reasearch and Technology Center, les versions 4.0 et 5.0 furent l'oeuvre du Wollongong Group, puis Performance Systems International et NYSERNet ont pris le relais et nous en sommes aujourd'hui à la version 7.0. ISODE a été entièrement écrit en langage C et fonctionne sous plusieurs systèmes d'exploitation, y compris Berkeley 4.4BSD UNIX et AT&T System V UNIX. C'est un produit gratuit et ouvert à tous les développeurs; cela signifie qu'on peut librement le copier, le modifier, le distribuer et l'utiliser pour un produit commercial.

Des expériences montrent que le développement des protocoles de niveau application exige autant ou plus d'effort que les protocoles de bas niveau. En produisant une implémentation gratuite et ouverte de protocoles OSI de niveaux supérieurs, on espère que les chercheurs dans l'enseignement, le public et les sociétés commerciales pourront commencer immédiatement à travailler sur des applications OSI. Il est à

remarquer que les responsables de développement de l'ISODE ne mésestiment pas TCP/IP, au contraire: ils le disent clairement dans l'introduction de [ISODE] et proposent des outils qui permettent l'interconnexion entre OSI et TCP/IP.

Voici les principaux outils de développement proposé par l'ISODE:

-Association Control Service (ACS)
-Remote Operations Service (ROS)
-Reliable Transfer Service (RTS)
-Presentation Service
-Presentation Protocol for TCP/IP
-Session Service
-Transport Service
-FTAM
-FTAM/FTP Gateway
-Virtual Terminal (VT)
-Directory System Agent (DSA) (" *Quipu*" library)
-*Pepsy* compiler

Parmi toutes ces implémentations, c'est le Presentation Service et le compilateur *pepsy* qui nous intéressent pour le développement de notre *X.435 API.*

Compilateur *pepsy*

Pepsy signifie *Presentation Element Parser and Structure-Generator (YACC-based).* C'est un outil récent qui remplace les anciens compilateurs *posy* et *pepy* de l'ISODE. *Pepsy* lit la description d'un module en ASN.1 et produit les deux choses suivantes:

1) Des structures C correspondantes avec plusieurs tables qui définissent le lien entre le module en ASN.1 et les structures C.

2) Un programme C comportant des fonctions qui encodent des structures C en encodage BER, décodent l'encodage BER en des structures C, ou impriment le contenu de l'encodage BER. (Cette dernière possibilité est très utile pour le debugging.)

Grâce à *pepsy*, nous avons résolu le problème de conversion entre le langage ASN.1 et l'encodage BER. *Pepsy* traduit la spécification ASN.1 du PDU Pedi en des structures C. Nous pouvons dès lors garnir les champs de ces structures C - éventuellement à l'aide des fonctions du Presentation Service de l'ISODE, puis encoder les structures C correctement garnies en encodage BER par les fonctions d'encodage générées par *pepsy.* Il ne reste alors plus qu'à passer le résultat de l'encodage BER au *X.400 Gateway API* pour l'envoyer à son destinataire. (Le décodage suit une démarche similaire.)

Ainsi, les fonctions de manipulation d'objet de notre *X.435 API* (voir 5.3.2) sont basées sur *pepsy.* En réalité, ces fonctions font simplement des manipulations sur les structures C générées par *pepsy.* Une fois que la structure correspondant à un message Pedi est correctement garnie, *pepsy* s'occupe du problème d'encodage et de

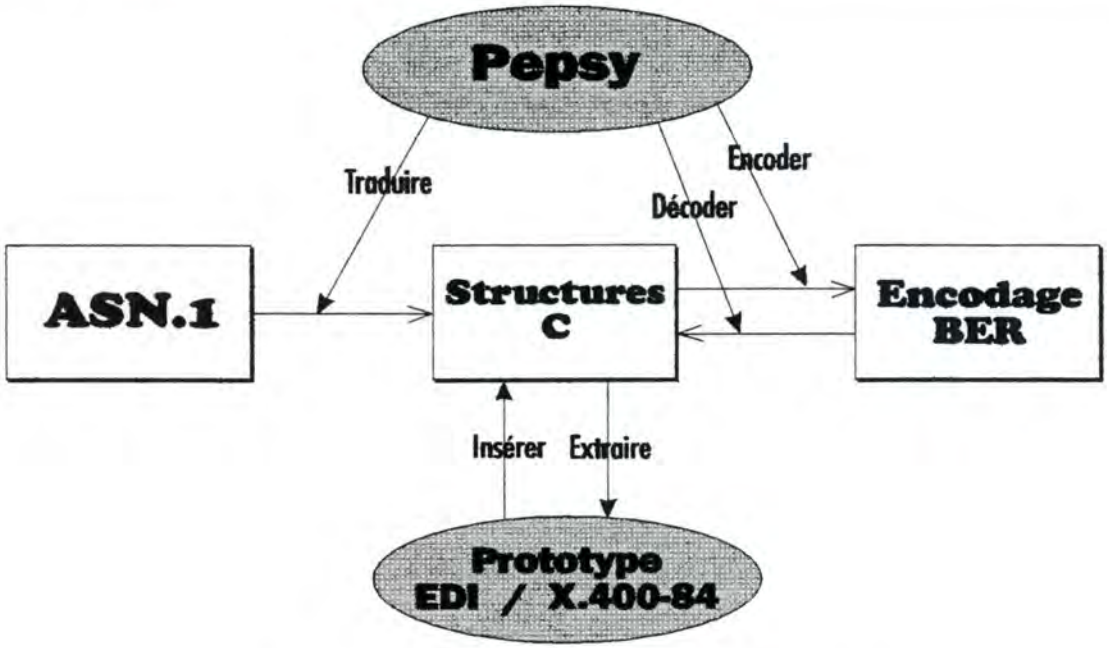

Figure 5-4 : Rôle de pepsy dans le prototype

décodage, nous déchargeant ainsi de ce travail fastidieux. (Voir figure 5-4)

5.4 Prototype X.435/X.400-84

Notre prototype X.435/X.400-84 est développé sur une station de travail SUN sous le système d'exploitation UNIX de SUN (SUN/OS) et est entièrement écrit en langage C. Les outils que nous utilisons (*X.400 Gateway API* et *pepsy*) ne supportant que le langage C, le choix de ce langage devient tout naturel, d'autant que c'est le langage universel en UNIX.

Notre prototype X.435/X.400-84 porte le nom ''XEDI'': cela signifie tout simplement *X.400-based EDI.*

> Note:
>
> Le terme ''XEDI'' sera dorénavant utilisé pour désigner le prototype X.435/X.400-84.

5.4.1 Fonctionnalités du prototype

Présentation

Le développement de XEDI est un projet de recherche. Ainsi, nous avons

volontairement délaissé le côté présentation pour nous consacrer uniquement au fond, c'est-à-dire à la mise en oeuvre du protocole Pedi. L'interface-utilisateur proposée est donc rudimentaire, il s'agit du strict minimum pour qu'un utilisateur puisse contrôler l'envoi et la réception des EDIMs.

> Note:
>
> Le terme ''utilisateur'' signifie ici la personne physique qui utilise XEDI sur un terminal. (On trouve des discussions sur le rôle de l'utilisateur dans 5.4.4)

Voici un exemple de ce que l'utilisateur peut visualiser à l'écran lors du lancement de XEDI:

```
**** Xedi version 1  (W. KAO, 1991, Brussels) ****

EDI Messages to send
--------------------

1) FILE_1.EDIFACT
2) FILE_2.EDIFACT

Incoming EDI Messages
---------------------

1) XEDI920510081423    sent by <originator's O/R Name>
2) XEDI920511124520    sent by <originator's O/R Name>
                       forwarded by <forwarding EDI user's O/R Name>
3) XEDI920511140305    sent by <originator's O/R Name>


1) Send  2) Accept  3) Refuse  4) Forward (R.N.A.)  5) Forward (R.A.)
6) View  7) Quit Xedi
Your choice :
```

Dans la rubrique ''EDI messages to send'' sont repris les noms des interchanges EDIFACT prêts à être envoyés. XEDI suppose que le EDI Name du destinataire se trouve déjà dans le champ ''Interchange Recipient'' du segment UNB de l'interchange. (Voir 1.2.6 pour le segment UNB)

Dans la rubrique ''Incoming EDI Messages'' sont repris les noms des EDIMs reçus par XEDI. Les noms sont assignés par XEDI, ils commencent par les lettres ''XEDI'' et la suite indique la date et l'heure d'arrivée. Les *O/R Names* de l'émetteur et des intermédiaires éventuels (ceux qui ont relayé le message) sont également affichés à l'écran.

XEDI affiche les actions possibles à l'utilisateur et attend une commande de celui-ci.

Contrôles des messages EDI

XEDI offre les possibilités de contrôle de messages qui sont prévues dans X.435:

1) Envoi d'un message *(Send)*

Cette action est la seule qui concerne la rubrique "EDI messages to send". L'utilisateur envoie un interchange à son destinataire après avoir entré les options liées au message. Les options possibles sont:

A. Permission au récepteur de passer la responsabilité
B. Demande de notification positive (PN)
C. Demande de notification négative (NN)
D. Demande de notification de relais (FN) (Note: cette option n'est possible que si l'option A a été requise)
E. Le nom du récepteur des notifications
F. Le date et l'heure d'expiration du message

X.435 prévoit d'autres options pour l'envoi d'un message: l'option "Related messages" permet à l'utilisateur de spécifier les noms des messages envoyés précédemment qui sont relatifs au message envoyé, et l'option "Obsoleted EDIMs" permet à l'utilisateur de spécifier les noms des messages envoyés précédemment qui sont rendus périmés par le message envoyé. Ces options ne sont pas obligatoires et nous ne les avons pas implémentées dans la version actuelle de XEDI.

2) Acceptation d'un message *(Accept)*

L'utilisateur accepte la responsabilité de l'EDIM reçu. Dans ce cas-ci, XEDI renvoie une notification positive (PN) à l'émetteur du message si cela avait été requis.

3) Refus d'un message *(Refuse)*

L'utilisateur refuse la responsabilité de l'EDIM reçu. Dans ce cas-ci, XEDI demande à l'utilisateur la raison du refus, rejette le message reçu et renvoie une notification négative (NN) à l'émetteur si cela avait été requis.

4) Relais d'un message sans acceptation de responsabilité *(Forward R.N.A.)*

L'utilisateur relaie la responsabilité de l'EDIM à un autre destinataire. Dans ce cas-ci, XEDI demande à l'utilisateur la raison du relais et renvoie une notification de relais (FN) à l'émetteur si cela avait été requis. Cette action n'est possible que si l'émetteur avait donné la permission au récepteur de passer la responsabilité. (Voir l'option A de l'action 1 ci-dessus)

5) Relais d'un message avec acceptation de responsabilité *(Forward R.A.)*

L'utilisateur accepte la responsabilité mais relaie l'EDIM à un autre destinataire. Dans ce cas-ci, XEDI demande à l'utilisateur la raison du relais et renvoie une

notification positive (PN) à l'émetteur si cela avait été requis.

6) Visualisation du message *(View)*

L'utilisateur visualise le contenu du message à l'écran. Cette action a l'air bien innocente, mais elle a des conséquences sur la responsabilité de l'EDIM. En effet, nous estimons que le fait de permettre à l'utilisateur de consulter le contenu d'EDIM reçu dévoile le secret et nous considérons que l'EDIM est ainsi *rendu disponible* à l'utilisateur. Dès lors, **après la visualisation du message, l'utilisateur ne peut plus le refuser ou le relayer à un autre destinataire sans accepter la responsabilité** (Actions 3 et 5).

Envoi et réception des notifications

Comme on peut lire ci-dessus, l'envoi des EDINs est géré par XEDI sans intervention de l'utilisateur. XEDI génère les bonnes EDINs au bon moment en accord avec la spécification de X.435.

La réception d'une EDIN est signalée à l'utilisateur par affichage à l'écran d'un message du genre:

> Negative Notification received for EDI message XEDI920509125512
> Initiated by UA or MS
> Basic code = 1
> Diagnostic code = 3

Le nom du message auquel l'EDIN fait référence est affiché à l'écran. Le niveau auquel l'EDIN a été générée est également indiqué; soit par l'utilisateur (ex: refus de l'utilisateur), soit par l'UA ou le MS (ex: message syntaxiquement incorrect), ou par le PDAU. Le ''basic code'' et le ''diagnostic code'' indiquent la raison de la génération de l'EDIN; on peut trouver la signification de ces codes dans [X.435].

Autres fonctionnalités

Quand l'utilisateur envoie un EDIM avec demande de notifications (PN, NN et FN), il espère bien évidemment que ces notifications lui arriveront dans un délai raisonnable. Mais il faut bien avertir l'utilisateur si un EDIM est resté longtemps sans notifications.

Ainsi, à chaque envoi d'EDIM avec demande de notifications, XEDI prend le soin de garder une copie de ce message. A chaque lancement de XEDI, si celui-ci détecte qu'un message est resté longtemps (disons 24 heures) sans notifications, alors un message d'avertissement est affiché à l'écran.

5.4.2 Architecture générale du prototype

XEDI est composé de deux applications et de plusieurs files (''queues'' en anglais). On trouve à la figure 5-5 la disposition de ces éléments.

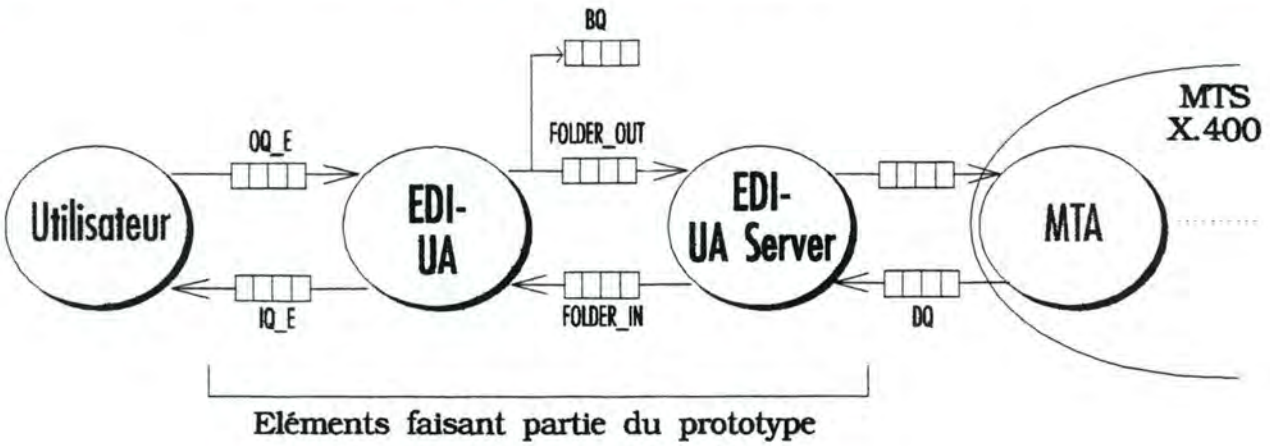

Figure 5-5 : Architecture générale du prototype

Les deux applications de XEDI ont des rôles distincts et travaillent à des niveaux différents. La première, l'EDI-UA, travaille au niveau X.435 (Pedi), alors que la seconde, l'EDI-UA Server, travaille au niveau X.400 (P1). Voici quelques mots sur les rôles des deux applications:

EDI-UA (EDI User Agent)

Le programme principal, qui doit être lancé par chaque utilisateur de XEDI. L'EDI-UA entre directement en contact avec l'utilisateur et permet à celui-ci d'envoyer, relayer, visualiser et recevoir des messages Pedi.

L'EDI-UA travaille uniquement au niveau X.435. Il prépare un message Pedi et le confie à EDI-UA Server dans le cas de l'envoi d'un interchange EDIFACT, et il extrait du message Pedi reçu de l'EDI-UA Server l'interchange EDIFACT et le confie à l'utilisateur dans le cas de la réception d'un message Pedi. Pour effectuer ces travaux, il utilise le *X.435 API* que nous avons créé.

EDI-UA Server

Le serveur central de XEDI. Il ne peut exister qu'un seul processus de l'EDI-UA Server sur le réseau local et il est lancé une fois pour toutes. EDI-UA Server a pour tâche principale de vérifier périodiquement (disons toutes les 30 secondes) si des messages Pedi sont à envoyer et si des messages P1 (contenant un message Pedi) sont à recevoir. Dans le premier cas il compose un message P1 et le confie au MTA; dans le second cas il en extrait le message Pedi et le confie à l'EDI-UA. Pour effectuer ces travaux, il utilise principalement le *X.400 Gateway API*, mais également le *X.435 API* dans le cas de l'envoi d'un message Pedi.

Cette dernière remarque peut paraître illogique. En effet, puisque l'EDI-UA Server travaille au niveau X.400, pourquoi aurait-il besoin du *X.435 API*? La raison est bien simple: lors de l'envoi d'un message Pedi, l'EDI-UA Server prépare l'enveloppe P1 sur base de l'enveloppe Pedi (le *heading*) du message à envoyer. Cela signifie que les valeurs des champs de l'enveloppe P1 sont déduites ou copiées de l'enveloppe Pedi.

Mais pour consulter les champs de l'enveloppe Pedi, il faut bien un outil qui puisse comprendre la structure d'un message Pedi (n'oublions pas qu'un message Pedi est présenté comme une simple suite d'octets à l'EDI-UA Server). D'où la nécessité d'utiliser le *X.435 API.*

Mais **la principale raison d'être de l'EDI-UA Server réside dans le fait qu'il joue également une partie du rôle du Message Store** : en effet, l'EDI-UA Server stocke tous les messages venant de l'extérieur dans des files (il y a une file par EDI-UA). De cette manière, **XEDI ne sera jamais débordé par une éventuelle arrivée massive de messages, même lorsque les EDI-UAs des utilisateurs sont inactifs** .

Voici les différentes files utilisées par XEDI:

IQ_E *(Input Queue Edifact)*

File servant à recevoir les interchanges EDIFACT venant de l'extérieur via l'EDI-UA. Elle garde ces interchanges qui sont éventuellement destinés à un programme d'application (ex: comptabilité) EDI.

OQ_E *(Output Queue Edifact)*

File servant à recevoir les interchanges EDIFACT générés par un programme d'application EDI. Elle garde ces interchanges en attendant que l'EDI-UA vienne les enlever.

DQ *(Delivery Queue)*

File géré par le MTA, qui y met les messages P1 venant de l'extérieur.

FOLDER_IN

File servant à recevoir les messages Pedi venant de l'extérieur via l'EDI-UA Server. L'EDI-UA Server prend les messages P1 de DQ, en extrait les messages Pedi et les place dans cette file.

FOLDER_OUT

File servant à recevoir les messages Pedi générés par l'EDI-UA. L'EDI-UA Server prend ces messages, compose des messages P1 pour les envoyer au MTA.

BQ *(Backup Queue)*

File servant à garder une copie des messages Pedi envoyés dans lesquels une demande de notifications a été requise. Cela a pour but de faire la corrélation entre les messages Pedi déjà envoyés et les notifications reçues ultérieurement.

5.4.3 Flux de messages

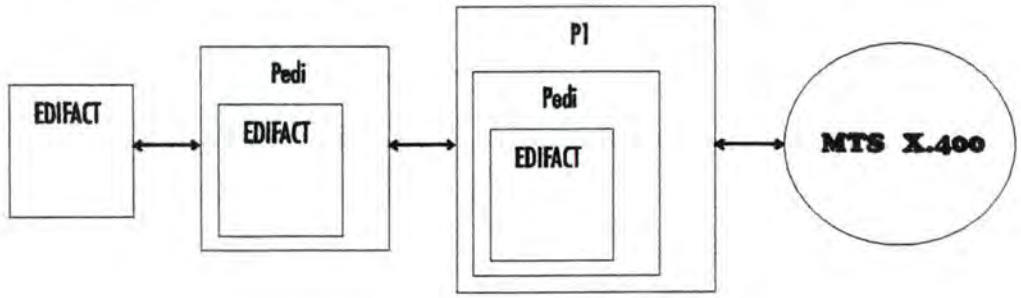

Figure 5-6 : Parcours d'un message EDI

Les messages EDI subissent plusieurs transformations dans XEDI. Nous allons voir quelles sont les différentes étapes de cette transformation dans le cas de l'envoi et de la réception d'un EDIM. (Voir figure 5-6)

L'envoi d'un message:

1) L'interchange EDIFACT à envoyer est mis dans OQ_E.

2) L'EDI-UA prend l'interchange EDIFACT de OQ_E, compose un message Pedi avec l'aide de l'utilisateur, puis le met dans FOLDER_OUT. S'il y a une demande de notifications, une copie du message Pedi est gardée dans BQ.

3) L'EDI-UA Server prend le message Pedi du FOLDER_OUT, compose un message P1, puis le confie au MTA.

La réception d'un message:

1) Le MTA met le message P1 reçu dans DQ.

2) L'EDI-UA Server prend le message P1 du DQ, en extrait le message Pedi, puis le met dans FOLDER_IN.

3) L'EDI-UA prend le message Pedi de FOLDER_IN, en extrait l'interchange EDIFACT, puis le met dans IQ_E.

Ainsi, un interchange EDIFACT reçoit successivement un en-tête Pedi *(heading)* et une enveloppe P1 avant d'être introduite dans le système de transfert de message (MTS) X.400.

5.4.4 Rôle de l'utilisateur

Le but de XEDI est simplement de vérifier la possibilité d'utiliser X.400-84 pour implémenter une application X.435. XEDI est avant tout un projet de recherche et non un produit commercial. Pour cette raison, nous avons volontairement fait certaines concessions sur la philosophie même d'un système de messagerie EDI. C'est notamment le cas du rôle de l'utilisateur (EDIMG-user).

Comme nous l'avons déjà précisé, le terme ''utilisateur'' employé dans ce chapitre signifie une personne physique. Ainsi, c'est cette personne qui devant son terminal accepte, refuse et relaie les EDIMs reçus. Mais tout cela semble bien illogique: de telles décisions ne devraient-elles pas être prises par un programme d'application (ex.: programme de comptabilité, de facturation, ...) en amont qui aurait préalablement consulté ses bases de données et effectué des traitements et des calculs? Le rôle de la personne dans XEDI paraît très discutable.

Il convient tout d'abord de bien faire la distinction entre le rôle de l'UA dans un système de messagerie interpersonnelle et le rôle de l'EDI-UA dans un système de messagerie EDI. L'UA est en quelque sorte le ''terminus'' d'un système de messagerie interpersonnelle, les messages qu'il reçoit sont destinés à des personnes physiques et il doit assister activement celles-ci dans la préparation d'un message interpersonnel. Normalement, il n'a pas de compte à rendre à une autre application. Le cas de l'EDI-UA est très différent: un EDI-UA reçoit des documents commerciaux, le traitement et le contrôle des ceux-ci ne sont pas du ressort d'une personne physique mais bien d'un programme d'application EDI. L'EDI-UA devrait être passif et se contenter de jouer le rôle de l'intermédiaire entre un programme d'application et le MTS X.400 sans intervention humaine. Un EDI-UA tout seul sans liaison avec un programme d'application EDI de niveau supérieur n'a en principe aucun sens.

Le choix de renforcer le rôle de la personne dans XEDI est voulu et forcé, car XEDI est pour l'instant un prototype indépendant et n'est lié à aucun autre programme d'application de niveau supérieur. Nous l'avons fait uniquement pour le besoin de tests de XEDI. La personne se permet de prendre des décisions à la place d'un programme d'application parce que ce dernier n'existe pas tout simplement. L'''utilisateur'' d'un système de messagerie EDI devrait être un programme d'application et non une personne physique. L'auteur de ce texte n'ignore pas que, pour en faire un produit commercial, il y a lieu de revoir la philosophie de XEDI.

5.4.5 Conformité du prototype

Quand le développement d'une application OSI se termine, les développeurs se posent inévitablement des questions concernant la conformité de leur application. En effet, les protocoles OSI de niveau Application sont si complexes et comportent tant de possibilités optionnelles que toute application OSI particulière a toujours beaucoup de peine à être parfaitement conforme avec la Recommandation OSI sur laquelle elle est basée.

Des outils de test de conformité existent sur le marché. L'organisation SPAG (Standard Promotion and Application Group) créée par un consortium de sociétés informatiques a pour mission de créer ces outils de test pour aider les développeurs à mettre au point leurs applications OSI et de promouvoir ainsi le modèle OSI. Ces outils testent l'interopérabilité de leurs systèmes avec une application OSI particulière pour déceler d'éventuelles erreurs de cette dernière. Mais pour le moment, ces outils ne reçoivent que peu de crédit de la part des développeurs d'applications OSI.

D'abord ces outils sont très onéreux: la location de matériel et de logiciels de test se chiffre par millions (voire dizaines de millions) de francs belges. Ensuite, rien ne prouve que ces outils aient prévu toutes les possibilités et toutes les situations possibles spécifiées dans la Recommandation OSI: des cas extrêmes peuvent très bien échapper aux contrôles de ces outils de test. Et puis, qu'est-ce qui prouve que ces outils sont eux-mêmes conformes, même en sachant qu'ils ont été conçus avec mille précautions? Nous nous trouvons dans un véritable cercle vicieux, et une application OSI qui passe avec succès le test n'est pas nécessairement conforme à 100% avec la Recommandation OSI sur laquelle elle est basée.

Actuellement, les développeurs ont plutôt tendance à faire communiquer l'application OSI qu'ils viennent de créer avec des applications déjà existantes sur le marché. On essaie de voir si l'application tient la route face aux produits concurrents et de déceler ainsi d'éventuels problèmes de conformité et de compatibilité. Cette méthode officieuse de test de conformité "par la pratique" peut paraître discutable, mais elle est une pratique courante faute de tests abordables et fiables à 100%. Pour information, la CEE vient d'acquérir le système de messagerie X.400 développé par la société BIM (Everberg, Belgique), alors que ce système n'a jamais été vérifié par un outil de test de conformité.

Concernant X.435, si CCITT avait exigé l'utilisation de MTA version 1988, alors une application X.435/X.400-84 aurait de toute manière été non conforme, car trop d'éléments de service - notamment ceux concernant la sécurité - sont manquants dans X.400-84 pour réaliser X.435 (voir 4.5). Mais puisque l'utilisation de MTA version 1984 est permise, une application EDI / X.400 peut certainement être conforme si elle est bien écrite, même en sachant qu'elle perd ainsi tous les avantages procurés par X.400-88 (sécurité, liste de distribution, ...).

Pour revenir à XEDI, ce prototype n'a pas non plus été vérifié par un outil de test de conformité. De toute manière, la parution de la Recommandation X.435 est si récente qu'il n'est même pas certain que des outils de test de conformité de X.435 existent déjà. Pour la même raison, nous n'avons pas pu faire communiquer XEDI avec d'autres applications X.435; car à notre connaissance, des applications commerciales de X.435 n'existent pas encore sur le marché à ce jour. Il est dès lors compréhensible que malgré tous nos efforts et toute notre attention, nous nous trouvions dans l'impossibilité de certifier la conformité de XEDI avec la Recommandation X.435. L'utilisation des APIs devrait réduire le risque de non-conformité, mais ce risque existe toujours malgré tout.

Tout ce qu'on peut affirmer c'est que XEDI ne devrait pas souffrir de grands problèmes de conformité, car à cause du délai imposé au projet, nous n'avons implémenté dans XEDI que le noyau de X.435, c'est-à-dire le strict minimum requis par la spécification de X.435. Nous devons cependant avouer que, à part les limitations dues à l'utilisation de la version 84 de X.400, certaines possibilités **optionnelles** ont été rejetées par manque de temps, mais cela est fait de manière à ne pas nuire à la conformité de XEDI. Ces possibilités sont:

1) Lors de l'envoi d'un EDIM, l'utilisateur devrait pouvoir utiliser l'option "Related messages" pour spécifier les noms des messages envoyés

précédemment qui sont relatifs au message envoyé, et l'option "Obsoleted EDIMs" pour spécifier les noms des messages envoyés précédemment qui sont rendus périmés par le message envoyé. Ces possibilités ne sont pas prévues dans la version actuelle de XEDI. (Ce problème a déjà été abordé dans 5.4.1)

2) X.435 prévoit la possibilité de joindre des informations supplémentaires sous quelque forme que ce soit (dessins, voix, ...) dans les *additional body parts* d'un EDIM. XEDI ne permet pas cette possibilité et s'il reçoit un EDIM comportant des informations supplémentaires, il ignore tout simplement ces informations.

3) De même, X.435 prévoit la possibilité pour l'utilisateur de retirer et d'ajouter des informations supplémentaires à un EDIM avant de le relayer à un autre destinataire si la responsabilité est acceptée. XEDI ne permet pas cette possibilité et l'utilisateur est obligé de relayer un EDIM tel quel, sans modification.

5.5 Problèmes rencontrés lors de l'implémentation du prototype

Au cours de l'implémentation de XEDI, nous avons rencontré des problèmes qui ont rendu le développement plus difficile. Ces problèmes sont souvent d'ordre pratique, mais également d'ordre conceptuel.

Tout d'abord, comme nous l'avons déjà indiqué, la définition du PDU X.435 telle qu'elle est présentée dans [X.435] contient des erreurs d'inattention et des références introuvables. Cela nous a imposé de longues heures de recherche et de lecture dans [X.435] et dans [X.400-88] pour constituer une version corrigée qu'on peut trouver dans l'Annexe B.

Ensuite, le *X.400 Gateway API* de la société RETIX comporte beaucoup d'erreurs. Cela est d'une telle gravité que l'auteur de ce texte estime avoir dépensé 40% de son temps à corriger ces erreurs, le reste du temps étant consacré au développement du *X.435 API* et de XEDI.

Mais le problème principal que nous avons rencontré est dû à une incompatibilité d'approche entre l'architecture de XEDI et la politique du dédoublement de champs de données dans un PDU X.435. Pour rappel, XEDI est divisé en deux applications, l'EDI-UA et l'EDI-UA Server. La première travaille au niveau X.435, tandis que la seconde travaille au niveau X.400. Lorsque l'EDI-UA confie un message Pedi à l'EDI-UA Server, ce dernier prépare l'enveloppe P1 en se basant sur l'enveloppe Pedi de l'EDIM. Cela est possible grâce au dédoublement de champs de données entre un PDU Pedi et un PDU P1: voyons ce que cela signifie.

Le PDU X.435 comporte plusieurs champs de données qui sont soit copiés de l'interchange EDIFACT, soit destinés à l'enveloppe P1. La raison de ce dédoublement est de permettre à l'EDI-UA de prendre certaines décisions relatives à l'interchange EDIFACT sans devoir connaître la syntaxe EDIFACT, et de préparer à

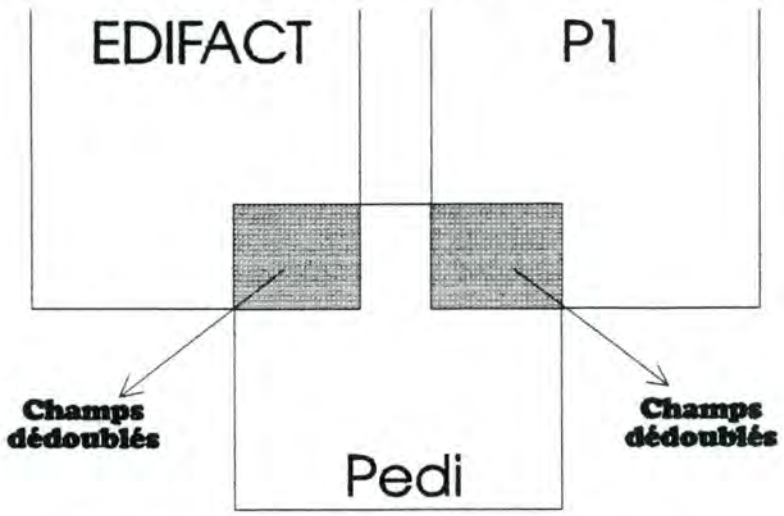

Figure 5-7 : Dédoublement de champs de données dans le message Pedi

l'avance l'enveloppe P1 sans devoir passer des informations au module de préparation de message P1 par un autre moyen. (Voir figure 5-7)

Nous avons trouvé cette idée très bonne. Ainsi, dans le cas de l'envoi d'un message Pedi, nous avons pris la décision de ne pas utiliser les services abstraits *OriginateEDIM* et *OriginateEDIN* de X.435 et de passer le message Pedi seul sans aucun autre paramètre au module de préparation du message P1 (EDI-UA Server). Nous pensions que, puisque les champs dédoublés permettaient la construction de l'enveloppe P1, il était inutile de passer d'autres paramètres.

Or, cette décision s'est avérée mauvaise. Car en ce qui concerne ces champs dédoublés, nous avons l'impression que X.435 n'a pas fait les choses jusqu'au bout. Pas de problème pour les champs de données dédoublés entre EDIFACT et Pedi, mais ce n'est malheureusement pas le cas pour ceux entre Pedi et P1. Ainsi, nous avons des situations où, bien que la décision de la valeur d'un champ de P1 soit du ressort de l'EDI-UA, celui-ci n'a pas le moyen de la communiquer à l'EDI-UA Server, car le champ en question n'est pas dédoublé. Citons deux exemples pour mieux comprendre ce problème:

1) X.435 spécifie que, lorsqu'on envoie un message P1 contenant une EDIN, le champ ''Priorité'' de l'enveloppe P1 doit être mis à la même valeur que celle de l'EDIM qui a provoqué cette notification. Or, le champ ''Priorité'' n'est pas dédoublé et il est donc impossible pour l'EDI-UA Server de déterminer la valeur de ce champ en consultant simplement l'EDIN à envoyer.

2) Le même problème se pose pour l' *O/R Name* du receveur de la notification lorsqu'on envoie un message P1 contenant une EDIN. Ce champ n'est pas dédoublé non plus et l'EDI-UA Server, en recevant un message Pedi contenant une notification, ignore tout à fait à qui l'envoyer.

Le problème ne se pose pas pour un système où la préparation des messages Pedi et P1 est regroupée dans un même module; mais dans le cas de XEDI, l'EDI-UA est

obligé de passer certaines autres informations à l'EDI-UA Server en plus du message Pedi à envoyer. En réalité, la file FOLDER_OUT décrite dans 5.4.2 reçoit non seulement les messages Pedi à envoyer, mais également des informations supplémentaires nécessaires à la préparation de l'enveloppe P1. Nous avons en quelque sorte été forcés de revenir aux services abstraits *OriginateEDIM* et *OriginateEDIN* de X.435.

Ce problème n'est pas dû à une erreur de conception de X.435. Les concepteurs de X.435 n'ont sans doute pas prévu la possibilité d'avoir une séparation complète de traitements entre X.435 et X.400, malgré la présence des champs dédoublés. En réalité, la cause du problème vient de l'architecture particulière de XEDI. En effet, nous aurions dû ne pas nous préoccuper de ces champs dédoublés et utiliser les services abstraits *OriginateEDIM* et *OriginateEDIN* de X.435. Ce problème a été décelé très tard dans le développement de XEDI et nous regrettons aujourd'hui d'avoir pris la décision de séparer les traitements de niveau X.435 et de niveau X.400.

Conclusion

Notre travail est ainsi achevé avec le développement de notre prototype X.435/X.400-84. Après avoir longuement analysé la faisabilité de l'utilisation de X.400-84 pour X.435, le prototype nous a permis de mettre en pratique le résultat de notre analyse.

Comme nous l'avons déjà dit, X.435 représente sans nul doute la solution d'avenir pour l'échange de document EDI. Dès que X.400-88 se sera généralisé, cette solution sera certainement adoptée par les partenaires commerciaux désireux d'entrer dans le monde EDI. Monsieur Ray Walker, le Vice-Président de l'UN/EDIFACT Board, estime que l'utilisation de X.400 pour l'EDI prendra son envol dès les années 1995-1996. Le rêve d'un monde EDI homogène, dans lequel tout le monde peut s'échanger librement des documents EDI sans aucun obstacle, ne serait finalement pas si lointain que cela.

Concernant X.400-84, si son utilisation pour X.435 est théoriquement possible, elle n'est certainement pas souhaitable à cause de l'absence des services de sécurité. Car contrairement au message interpersonnel, un message EDI est un document commercial et peut souvent être confidentiel et d'une grande importance. Sans la protection des services de sécurité, on n'a aucun moyen de se protéger contre une éventuelle divulgation non autorisée ou falsification du message EDI. C'est un défaut grave qui a de lourdes conséquences sur le plan pratique et juridique. A côté de cela, l'absence du *Message Store*, l'éventuelle difficulté d'intégration de X.500 et les autres éléments de service manquants de X.435/X.400-84 paraissent bien dérisoires.

X.435/X.400-84 n'est qu'une solution intérimaire, en attendant que des outils de développement de X.400-88 connus et fiables soient disponibles dans le monde informatique. A cause de l'absence des services de sécurité, nous pensons qu'il s'agit d'une solution qui risque de ne jamais voir le jour, bien que sa conformité par rapport à la Recommandation X.435 soit malgré tout satisfaisante. Notre prototype X.435/X.400-84 a donc une valeur plus académique que pratique, nous le concédons volontiers. Au moins, nous espérons que ce texte aura le mérite de mettre en lumière l'importance des services de sécurité dans un réseau de télécommunication.

Bibliographie

[BLOCH]
Bloch S. :
EDI : Echange de Données Informatisé - Tome 1
Introduction à l'Echange de Données Structurées en Syntaxe EDIFACT
Eyrolles, 1991, Paris

[DELEARD]
Deléard D. :
EDI: la Mise en Place
Revue "Télécoms", juillet/août 1991, Paris

[F.435]
CCITT :
Message Handling Systems: EDI Messaging Service
Draft Recommendations F.435 (Version 5.0)
CCITT, 1990, Lausanne

[GENILLOUD]
Genilloud G. :
X.400 for EDI Communications
Swiss Federal Institute of Technology, 1988, Lausanne

[GH]
Gifkins M. & Hitchcock D. :
The EDI Handbook - Trading in the 1990s
Blenheim Online, 1988, London

[GIFKINS]
Gifkins M. :
EDI Technology
Blenheim Online, 1989, London

[HILL]
Hill R. :
EDI and X.400 using Pedi
The Guide for Implementors and Users
Technology Appraisals, 1990, Middlesex (UK)

[HS]
Henshall J. & Shaw S.:
OSI Explained: End-to-End Computer Communication Standard
Ellis Horwood Limited, 1988, Chichester (UK)

[ISODE]
Rose M., Onions J. & Robbins C. :
The ISO Development: User's Manual (Version 7)
Performance Systems International Inc, 1991, USA

[ISO-9735]
ISO :
EDIFACT: Application Level Syntax Rules
1988, Genève

[KENNY]
Kenny P. :
EDI and X.400 - Implementation Issues
COMPAT '89 Conference, 1989, Munich

[MK]
Mehnen H. & Kron H-P. :
An Introduction to EDIFACT Advanced Implementation
COMPAT '89 Conference, 1989, Munich

[RETIX]
Retix :
MH-445 Programmer Guide - Gateway API Library
Retix, 1990, Santa Monica

[UNTDID]
United Nations Economic Commission for Europe:
United Nations Trade Data Interchange Directory (UNTDID)
Issue 90.1, 1990, Genève

[VANBAST]
Van Bastelaer Ph. :
Introduction Technique à l'Echange de Données Informatisé (EDI)
(Notes du cours ''Conception de Systèmes Informatiques'')
Institut d'Informatique, FUNDP, 1991, Namur

[VANGUARD]
Vanguard :
EDI and X.400 Study
Department of Trade and Industry (UK), 1988, UK

[VANOOST]
Van Oost S. :
The Use of CCITT X.400 Recommendations for EDI
(Mémoire de fin d'étude)
Institut d'Informatique, FUNDP, 1991, Namur

[X.208]
CCITT :
Data Communication Networks - Specification of Abstract Syntax Notation One (ASN.1)
Recommendation X.208
CCITT, 1988, Melbourne

[X.209]
CCITT :
Data Communication Networks - Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1)
Recommendation X.209
CCITT, 1988, Melbourne

[X.400-API]
X.400 API Association :
X.400 Gateway API Specification
X.400 API Association, 1989, Herndon (USA)

[X.400-84]
CCITT :
Data Communication Networks - Message Handling Systems
Recommendations X.400 - X.430
CCITT, 1984, Malaga-Torremolinos

[X.400-88]
CCITT :
Data Communication Networks - Message Handling Systems
Recommendations X.400 - X.420
CCITT, 1988, Melbourne

[X.435]
CCITT :
Message Handling Systems: EDI Messaging System
Draft Recommendations X.435 (Version 5.0)
CCITT, 1990, Lausanne

[X.500]
CCITT :
The Directory - Overview of Concept, Models and Services
Recommendations X.500
CCITT, 1988, Melbourne

[X.509]
CCITT :
The Directory - Authentication Framework
Recommendations X.509
CCITT, 1988, Melbourne

Annexe A : Liste des erreurs du PDU Pedi

Cette liste se rapporte au document suivant:

Message Handling Systems: EDI Messaging System
Draft Recommendation X.435
Version 5.0, June 15, 1990, Lausanne

La définition en ASN.1 du PDU X.435, telle qu'elle est présentée dans l'Annexe B du document cité ci-dessus, comporte les erreurs et les références introuvables suivantes:

Page 104

```
EDIApplicationSecurityExtension ::= ExtensionsField
```

doit être remplacé par

```
EDIApplicationSecurityExtension ::= ExtensionField
```

Page 111

```
FNReasonCodeField ::= CHOICE {
          .
          .
     fn-pdau-reason-code          [2] FNPDAUReasonCodeFields }
```

doit être remplacé par

```
FNReasonCodeField ::= CHOICE {
          .
          .
     fn-pdau-reason-code          [2] FNPDAUReasonCodeField }
```

Page 112

```
FNUserBasicCodeField ::= INTEGER {
          .
          .
     interchange-sender-unknown (9),
     interchange-sender-unknown (10),
          .
          .
     } (0..ub-reason-code)
```

doit être remplacé par

```
FNUserBasicCodeField ::= INTEGER {
          .
          .
```

```
    interchange-sender-unknown (9),
    interchange-recipient-unknown (10),
        .
        .
    } (0..ub-reason-code)
```

Page 107

Dans la définition

```
SecurityElementsField ::= SEQUENCE {
    original-content        [0] Content OPTIONAL,
        .
        .
    }
```

le type *Content* ne se trouve nulle part, ni dans X.435, ni parmi les types importés par X.435. Il est en réalité défini dans X.411 sous la forme

```
Content ::= OCTET STRING
```

Il faut donc ajouter *Content* dans la liste des types importés de X.411 par X.435 et faire la modification suivante à la page 99:

```
--MTS Abstract Service
    MessageDeliveryTime, ....., ContentIntegrityCheck, Content
        FROM MTSAbstractService {.....}
```

Annexe B : PDU Pedi (Version corrigée)

```
--*****************************************************************************
--* CCITT X.435 Recommendation Pedi Protocol Data Unit                        *
--* Edited by Wei-Chao KAO              Nov. 1991, BIM                        *
--*                                                                           *
--* Cette version de PDU Pedi tient compte des erreurs                        *
--* de la Recommandation X.435  (version 5.0)                                 *
--*****************************************************************************


--Begin Pedi definitions ***********************************************

EDIMSInformationObjects {joint-iso-ccitt mhs-motis(6) edims(7) modules(0) information-objects(2)}

DEFINITIONS IMPLICIT TAGS ::=
BEGIN

InformationObject ::= CHOICE
        {
        edim    [0] EDIM,
        edin    [1] EDIN
        }

EDIMIdentifier ::= SET
        {
        user    [0] ORName,
        user-relative-identifier        [1] LocalReference
        }

LocalReference ::= PrintableString(SIZE(0..64))  --ub-local-reference

ExtensionField ::= SEQUENCE
        {
        type    [0] EDIM-EXTENSION,
        criticality     [1] Criticality DEFAULT FALSE,
        value   [2] ANY DEFINED BY type DEFAULT NULL NULL
        }

Criticality ::= BOOLEAN

EDIM-EXTENSION MACRO ::=
        BEGIN
        TYPE NOTATION ::= DataType Critical | empty
        VALUE NOTATION ::= value(VALUE OBJECT IDENTIFIER)
        DataType ::= type (X) Default | empty
        Default ::= ''DEFAULT'' value (X) | empty
        Critical ::= ''CRITICAL'' | empty
        END

EDIM ::= SEQUENCE
        {
        heading Heading,
        body    Body
        }

IdentificationCode ::= T61String(SIZE(1..35))  --ub-identification-code

IdentificationCodeQualifier ::= T61String(SIZE(1..4))  --ub-identification-code-qualifier

RoutingAddress ::= T61String(SIZE(1..14))  --ub-routing-address

Heading ::= SEQUENCE
        {
        this-EDIM       [1] ThisEDIMField,
        originator      [2] OriginatorField OPTIONAL,
        recipients      [3] RecipientsField OPTIONAL,
        edin-receiver   [4] EDINReceiverField OPTIONAL,
        responsability-forwarded        [5] ResponsabilityForwarded DEFAULT FALSE,
        edi-bodypart-type       [6] EDIBodyPartType DEFAULT {joint-iso-ccitt mhs-motis(6) edims(7) id-bp(11) id-edifact-
ISO646(0)},
        incomplete-copy [7] IncompleteCopyField DEFAULT FALSE,
        expiry-time     [8] ExpiryTimeField OPTIONAL,
        related-messages        [9] RelatedMessagesField OPTIONAL,
        obsoleted-EDIMs [10] ObsoletedEDIMsField OPTIONAL,
        edi-application-security-elements       [11] EDIApplicationSecurityElementsField OPTIONAL,
        cross-referencing-information   [12] CrossReferencingInformationField OPTIONAL,

        --Begin Fields from EDIFACT Interchange
        edi-message-type        [13] EDIMessageTypeField OPTIONAL,
        service-string-advice   [14] ServiceStringAdviceField OPTIONAL,
        syntax-identifier       [15] SyntaxIdentifierField OPTIONAL,
        interchange-sender      [16] InterchangeSenderField OPTIONAL,
        date-and-time-of-preparation    [17] DateAndTimeOfPreparationField OPTIONAL,
        application-reference   [18] ApplicationReferenceField OPTIONAL
        --End Fields from EDIFACT

        heading-extensions      [19] HeadingExtensionsField OPTIONAL
        }


ThisEDIMField ::= EDIMIdentifier

OriginatorField ::= ORName

RecipientsField ::= SET OF RecipientsSubField
```

```
RecipientsSubField ::= SEQUENCE
        {
        recipient       [1] RecipientField,
        action-request  [2] ActionRequestField DEFAULT {joint-iso-ccitt mhs-motis(6) edims(7) id-for(13) id-for-action(0)},
        edi-notification-requests-field [3] EDINotificationRequestsField OPTIONAL,
        responsability-passing-allowed  [4] ResponsabilityPassingAllowedField DEFAULT FALSE,

        --Begin Fields from EDIFACT UNB
        interchange-recipient   [5] InterchangeRecipientField OPTIONAL,
        recipient-reference     [6] RecipientReferenceField OPTIONAL,
        interchange-control-reference   [7] InterchangeControlReferenceFiled OPTIONAL,
        processing-priority-code        [8] ProcessingPriorityCodeField OPTIONAL,
        acknowledgement-request [9] AcknowledgementRequestField DEFAULT FALSE,
        communications-agreement-id     [10] CommunicationsAgreementIdField OPTIONAL,
        test-indicator  [11] TestIndicatorField DEFAULT FALSE,
        --End Fileds from EDIFACT UNB

        --Begin Fields from ANSIX12 ISA
        authorization-information       [12] AuthorizationInformationField OPTIONAL
        --End Fields from ANSIX12 ISA

        recipient-extensions    [13] RecipientExtensionsField OPTIONAL
        }

RecipientField ::= ORName

ActionRequestField ::= OBJECT IDENTIFIER

EDINotificationRequestsField ::= SEQUENCE
        {
        edi-notification-requests       [0] EDINotificationRequests DEFAULT {},
        edi-notification-security       [1] EDINotificationSecurity DEFAULT {},
        edi-reception-security  [2] EDIReceptionSecurity DEFAULT {}
        }

EDINotificationRequests ::= BIT STRING
        {
        pn (0),
        nn (1),
        fn (2)
        } (SIZE(0..16))  --ub-bit-options

EDINotificationSecurity ::= BIT STRING
        {
        proof (0),
        non-repudiation (1)
        } (SIZE(0..16))  --ub-bit-options

EDIReceptionSecurity ::= BIT STRING
        {
        proof (0),
        non-repudiation (1)
        } (SIZE(0..16))  --ub-bit-options

InterchangeRecipientField ::= SEQUENCE
        {
        recipient-identification-code   [0] IdentificationCode,
        identification-code-qualifier   [1] IdentificationCodeQualifier OPTIONAL,
        routing-address [2] RoutingAddress OPTIONAL
        }

RecipientReferenceField ::= SEQUENCE
        {
        recipient-reference     [0] RecipientReference,
        recipient-reference-qualifier   [1] RecipientReferenceQualifier OPTIONAL
        }

RecipientReference ::= T61String(SIZE(1..14))  --ub-recipient-reference

RecipientReferenceQualifier ::= T61String(SIZE(1..2))  --ub-recipient-reference-qualifier

RecipientExtensionsField ::= SET OF RecipientExtensionsSubField

RecipientExtensionsSubField ::= ExtensionField

EDINReceiverField ::= SEQUENCE
        {
        edin-receiver-name      [0] ORName,
        original-edim-identifier        [1] EDIMIdentifier OPTIONAL,
        first-recipient [2] FirstRecipientField OPTIONAL
        }

ResponsabilityForwarded ::= BOOLEAN  --default FALSE

EDIBodyPartType ::= OBJECT IDENTIFIER  --default EDIFACT-ISO646

EDIMessageTypeField ::= SET OF EDIMessageTypeFieldSubField

EDIMessageTypeFieldSubField ::= T61String(SIZE(1..ub-6))  --edi-message-type

ResponsabilityPassingAllowedField ::= BOOLEAN  --Default FALSE

IncompleteCopyField ::= BOOLEAN  --Default FALSE

ExpiryTimeField ::= UTCTime

RelatedMessagesField ::= SEQUENCE OF RelatedMessageReference
```

```
RelatedMessageReference ::= CHOICE
        {
        edi-message-reference   [0] EDIMIdentifier,
        external-message-reference      [1] ExternalMessageReference
        }

ExternalMessageReference ::= EXTERNAL

ObsoletedEDIMsField ::= SEQUENCE OF ObsoletedEDIMsSubfiled

ObsoletedEDIMsSubfiled ::= EDIMIdentifier

EDIApplicationSecurityElementsField ::= SEQUENCE
        {
        edi-application-security-element        [0] EDIApplicationSecurityElement OPTIONAL,
        edi-encrypted-primary-bodypart  [1] BOOLEAN OPTIONAL
        edi-application-security-extensions     [2] EDIApplicationSecurityExtensions OPTIONAL
        }

EDIApplicationSecurityElement ::= BIT STRING(SIZE(0..8191))  --ub-edi-application-security-elements

EDIApplicationSecurityExtensions ::= SEQUENCE OF EDIApplicationSecurityExtension

EDIApplicationSecurityExtension ::= ExtensionField

CrossReferencingInformationField ::= SET OF CrossReferencingInformationSubField

CrossReferencingInformationSubField ::= SEQUENCE
        {
        application-cross-reference     [0] ApplicationCrossReference,
        message-reference       [1] MessageReference OPTIONAL,
        body-part-reference     [2] BodyPartReference
        }

ApplicationCrossReference ::= OCTET STRING

MessageReference ::= EDIMIdentifier

ServiceStringAdviceField ::= SEQUENCE
        {
        component-data-element-separator        [0] ComponentDataElementSeparator,
        data-element-separator  [1] DataElementSeparator,
        decimal-notation        [2] DecimalNotation,
        release-indicator       [3] ReleaseIndicator OPTIONAL,
        reserved        [4] Reserved OPTIONAL,
        segment-terminator      [5] SegmentTerminator
        }

ComponentDataElementSeparator ::= OCTET STRING(SIZE(1))

DataElementSeparator ::= OCTET STRING(SIZE(1))

DecimalNotation ::= OCTET STRING(SIZE(1))

ReleaseIndicator ::= OCTET STRING(SIZE(1))

Reserved ::= OCTET STRING(SIZE(1))

SegmentTerminator ::= OCTET STRING(SIZE(1))

SyntaxIdentifierField ::= SEQUENCE
        {
        syntax-identifier       SyntaxIdentifier,
        syntax-version  SyntaxVersion
        }

SyntaxIdentifier ::= T61String(SIZE(1..4))  --ub-syntax-identifier

SyntaxVersion ::= PrintableString (SIZE(1..4))  --ub-syntax-version

InterchangeSenderField ::= SEQUENCE
        {
        sender-identification   [0] IdentificationCode,
        identification-code-qualifier   [1] IdentificationCodeQualifier OPTIONAL,
        address-for-reverse-routing     [2] RoutingAddress OPTIONAL  --EDIFACT Routing Information
        }

DateAndTimeOfPreparationField ::= UTCTime

InterchangeControlReferenceFiled ::= T61String(SIZE(1..14))  --ub-interchange-control-reference

ApplicationReferenceField ::= T61String(SIZE(1..14))  --ub-application-reference

ProcessingPriorityCodeField ::= T61String(SIZE(1..1))  --ub-processing-priority-code

AcknowledgementRequestField ::= BOOLEAN  --default FALSE

CommunicationsAgreementIdField ::= T61String(SIZE(1..35))  --ub-communications-agreement-id

TestIndicatorField ::= BOOLEAN

AuthorizationInformationField ::= SEQUENCE
        {
        authorization-information       [0] AuthorizationInformation,
        authorization-information-qualifier     [1] AuthorizationInformationQualifier OPTIONAL
        }

AuthorizationInformation ::= T61String(SIZE(1..10))  --ub-authorization-information
```

```
AuthorizationInformationQualifier ::= T61String(SIZE(1..2))  --ub-authorization-information-qualifier

HeadingExtensionsField ::= SET OF HeadingExtensionsSubField

HeadingExtensionsSubField ::= ExtensionField


Body ::= SEQUENCE
        {
        primary-body-part       PrimaryBodyPart,
        additional-body-parts   OtherBodyParts OPTIONAL
        }

PrimaryBodyPart ::= CHOICE
        {
        edi-body-part   [0] EDIBodyPart,
        forwarded-EDIM  [1] EDIMBodyPart
        }

OtherBodyParts ::= SEQUENCE OF EDIM-ExternallyDefinedBodyPart

EDIBodyPart ::= OCTET STRING

EDIMBodyPart ::= SEQUENCE
        {
        parameters      [0] MessageParameters OPTIONAL,
        data    [1] MessageData
        }

MessageParameters ::= SET
        {
        delivery-time   [0] MessageDeliveryTime OPTIONAL,
        delivery-envelope       [1] OtherMessageDeliveryFields OPTIONAL,
        other-parameters        [2] EDISupplementaryInformation OPTIONAL
        }

MessageData ::= SEQUENCE
        {
        heading Heading,
        body    BodyOrRemoved
        }

BodyOrRemoved ::= SEQUENCE
        {
        primary-or-removed      PrimaryOrRemoved,
        additional-body-parts   AdditionalBodyParts OPTIONAL
        }

PrimaryOrRemoved ::= CHOICE
        {
        removed-edi-body        [0] NULL,
        primary-body-part       [1] EXPLICIT PrimaryBodyPart
        --Recursive definition: see above for PrimaryBodyPart definition
        }

AdditionalBodyParts ::= SEQUENCE OF CHOICE
        {
        external-body-part      [0] EDIM-ExternallyDefinedBodyPart,
        place-holder    [1] BodyPartPlaceHolder
        }

BodyPartPlaceHolder ::= EDIM-ExternallyDefinedBodyPart

EDIM-ExternallyDefinedBodyPart ::= SEQUENCE
        {
        body-part-reference     [0] BodyPartReference OPTIONAL,
        external-body-part      [1] ExternallyDefinedBodyPart
        }

BodyPartReference ::= INTEGER  --unique within an EDIM

EDISupplementaryInformation ::= IA5String(SIZE(1..256))  --ub-supplementary-info-length


EDIN ::= CHOICE
        {
        positive-notification   [0] PositiveNotificationFields,
        negative-notification   [1] NegativeNotificationFields,
        forwarded-notification  [2] ForwardedNotificationFields
        }

PN ::= EDIN

NN ::= EDIN

FN ::= EDIN

CommonFields ::= SEQUENCE
        {
        subject-edim    [1] SubjectEDIMField,
        edin-originator [2] EDINOriginatorField,
        first-recipient [3] FirstRecipientField OPTIONAL,
        notification-time       [4] NotificationTimeField,
        notification-security-elements  [5] SecurityElementsField OPTIONAL,
        edin-initiator  [6] EDINInitiatorField
        notifications-extensions        [7] NotificationExtensionsField OPTIONAL
        }
```

```
SubjectEDIMField ::= EDIMIdentifier

EDINOriginatorField ::= ORName

FirstRecipientField ::= ORName

NotificationTimeField ::= UTCTime

SecurityElementsField ::= SEQUENCE
        {
        original-content        [0] Content OPTIONAL,
        original-content-integrity-check        [1] ContentIntegrityCheck OPTIONAL,
        edi-application-security-elements       [2] EDIApplicationSecurityElementsField OPTIONAL
        security-extensions     [3] ::= SecurityExtensionsField OPTIONAL
        }

SecurityExtensionsField ::=SET OF SecurityExtensionsSubField

SecurityExtensionsSubField ::= ExtensionField

EDINInitiatorField ::= ENUMERATED
        {
        internal-ua (0),
        external-ua (1),
        internal-ms (2)
        }

NotificationExtensionsField ::= SET OF NotificationExtensionsSubField

NotificationExtensionsSubField ::= ExtensionField

PositiveNotificationFields ::= SEQUENCE
        {
        pn-common-fields        [0] CommonFields,
        pn-supplementary-information    [1] EDISupplementaryInformation OPTIONAL
        pn-extensions   [2] PNExtensionsField OPTIONAL
        }

PNExtensionsField ::= SET OF PNExtensionSubField

PNExtensionSubField ::= ExtensionField

NegativeNotificationFields ::= SEQUENCE
        {
        nn-common-fields        [0] CommonFields,
        nn-reason-code  [1] NNReasonCodeField,
        nn-supplementary-information    [2] EDISupplementaryInformation OPTIONAL
        nn-extensions   [3] NNExtensionsField OPTIONAL
        }

NNReasonCodeField ::= CHOICE
        {
        nn-ua-ms-reason-code    [0] NNUAMSReasonCodeField,
        nn-user-reason-code     [1] NNUserReasonCodeField,
        nn-pdau-reason-code     [2] NNPDAUReasonCodeField
        }

NNUAMSReasonCodeField ::= SEQUENCE
        {
        nn-ua-ms-basic-code     [0] NNUAMSBasicCodeField,
        nn-ua-ms-diagnostic     [1] NNUAMSDiagnosticField OPTIONAL
        }

NNUAMSBasicCodeField ::= INTEGER
        {
        unspecified (0),
        cannot-deliver-to-user (1),
        delivery-timeout (2),
        message-discarded (3),
        subscription-terminated (4),
        forwarding-error (5),
        security-error (6)
        } (0..32767)  --ub-reason-code

NNUAMSDiagnosticField ::= INTEGER
        {
        --General diagnostic codes
        protocol-violation (1),
        edim-originator-unknown (2),
        edim-recipient-unknown (3),
        edim-recipient-ambiguous (4),
        action-request-not-supported (5),
        edim-expired (6),
        edim-obsoleted (7),
        duplicate-edim (8),
        unsupported-extension (9),
        incomplete-copy-rejected (10),
        edim-too-large-for-application (11),
        forwarded-edim-not-delivered (12),
        forwarded-edim-delivered-time-out (13),
        forwarded-loop-detected (14),
        unable-to-accept-responsability (15),
        --Interchange header diagnostic codes
        interchange-sender-unknown (16),
        interchange-recipient-unknown (17),
        invalid-heading-field (18),
        invalid-bodypart-type (19),
        invalid-message-type (20),
```

```
        invalid-syntax-id (21),
        --Security error diagnostic codes
        message-integrity-failure (22),
        forwarded-message-integrity-failure (23),
        unsupported-algorithm (24),
        decryption-failed (25),
        token-error (26),
        unable-to-sign-notification (27),
        unable-to-sign-message-receipt (28),
        authentication-failure (29),
        security-context-failure (30),
        message-sequence-failure (31),
        message-security-labelling-failure (32),
        repudiation-failure (33),
        proof-of-failure (34)
        } (1..32767)  --ub-reason-code

NNUserReasonCodeField ::= SEQUENCE
        {
        nn-user-basic-code      [0] NNUserBasicCodeField,
        nn-user-diagnostic      [1] NNUserDiagnosticField OPTIONAL
        }

NNUserBasicCodeField ::= INTEGER
        {
        unspecified (0),
        syntax-error (1),
        interchange-sender-unknown (2),
        interchange-recipient-unknown (3),
        invalid-heading-field (4),
        invalid-body-part-type (5),
        invalid-message-type (6),
        functional-groups-not-supported (7),
        subscription-terminated (8),
        no-bilateral-agreement (9),
        user-defined-reason (10)
        } (0..32767)  --ub-reason-code

NNUserDiagnosticField ::= INTEGER

NNPDAUReasonCodeField ::= SEQUENCE
        {
        nn-pdau-basic-code      [0] NNPDAUBasicCodeField,
        nn-pdau-diagnostic      [1] NNPDAUDiagnosticField OPTIONAL
        }

NNPDAUBasicCodeField ::= INTEGER
        {
        unspecified (0),
        undeliverable-mail (1),
        physical-rendition-not-performed (2)
        } (0..32767)  --ub-reason-code

NNPDAUDiagnosticField ::= INTEGER
        {
        undeliverable-mail-physical-delivery-address-incorrect (1),
        undeliverable-mail-physical-delivery-office-incorrect-or-invalid (2),
        undeliverable-mail-physical-delivery-address-incomplete (3),
        undeliverable-mail-recipient-unknown (4),
        undeliverable-mail-recipient-deceased (5),
        undeliverable-mail-recipient-refused-to-accept (6),
        undeliverable-mail-organization-expired (7),
        undeliverable-mail-recipient-did-not-claim (8),
        undeliverable-mail-recipient-changed-address-permanently (9),
        undeliverable-mail-recipient-changed-address-temporarily (10),
        undeliverable-mail-recipient-changed-temporary-address (11),
        undeliverable-mail-new-address-unknown (12),
        undeliverable-mail-recipient-did-not-want-forwarding (13),
        undeliverable-mail-originator-prohibited-forwarding (14),
        physical-rendition-attributes-not-supported (15)
        } (1..32767)  --ub-reason-code

NNExtensionsField ::= SET OF NNExtensionsSubField

NNExtensionsSubField ::= ExtensionField

ForwardedNotificationFields ::= SEQUENCE
        {
        fn-common-fields        [0] CommonFields,
        forwarded-to    [1] ForwardedTo,
        fn-reason-code  [2] FNReasonCodeField,
        fn-supplementary-information    [3] EDISupplementaryInformation OPTIONAL
        fn-extensions   [4] FNExtensionsField OPTIONAL
        }

ForwardedTo ::= ORName

FNReasonCodeField ::= CHOICE
        {
        fn-ua-ms-reason-code    [0] FNUAMSReasonCodeField,
        fn-user-reason-code     [1] FNUserReasonCodeField,
        fn-pdau-reason-code     [2] FNPDAUReasonCodeField
        }

FNUAMSReasonCodeField ::= SEQUENCE
        {
        fn-ua-ms-basic-code     [0] FNUAMSBasicCodeField,
        fn-ua-ms-diagnostic     [1] FNUAMSDiagnosticField OPTIONAL,
        fn-security-check       [2] FNUAMSSecurityCheckField DEFAULT FALSE
```

```
        }

FNUAMSBasicCodeField ::= INTEGER
        {
        unspecified (0),
        onward-routing (1),
        recipient-unknown (2),
        originator-unknown (3),
        forwarded-by-edi-ms (4)
        } (0..32767)  --ub-reason-code

FNUAMSDiagnosticField ::= INTEGER
        {
        recipient-name-changed (0),
        recipient-name-deleted (1)
        } (0..32767)  --ub-reason-code

FNUAMSSecurityCheckField ::= BOOLEAN

FNUserReasonCodeField ::= SEQUENCE
        {
        fn-user-basic-code      [0] FNUserBasicCodeField,
        fn-user-diagnostic      [1] FNUserDiagnosticField OPTIONAL
        }

FNUserBasicCodeField ::= INTEGER
        {
        unspecified (0),
        forwarded-for-archiving (1),
        forwarded-for-information (2),
        forwarded-for-additional-action (3),
        subscription-changed (4),
        heading-field-not-supported (5),
        bodypart-type-not-supported (6),
        message-type-not-supported (7),
        syntax-identifier-not-supported (8),
        interchange-sender-unknown (9),
        interchange-recipient-unknown (10),
        user-defined-reason (11)
        } (0..32767)  --ub-reason-code

FNUserDiagnosticField ::= INTEGER

FNPDAUReasonCodeField ::= SEQUENCE
        {
        fn-pdau-basic-code      [0] FNPDAUBasicCodeField,
        fn-pdau-diagnostic      [1] FNPDAUDiagnosticField OPTIONAL
        }

FNPDAUBasicCodeField ::= INTEGER
        {
        unspecified (0),
        forwarded-for-physical-rendition-and-delivery (1)
        } (0..32767)  --ub-reason-code

FNPDAUDiagnosticField ::= INTEGER

FNExtensionsField ::= SET OF FNExtensionsSubField

FNExtensionsSubField ::= ExtensionField

--End Pedi definitions*****************************************************



--Begin Imported definitions (from X.411, X.420, X.509) ***********************
--These definitions do not form an integral part of X.435 but are used by X.435

MessageDeliveryTime ::= Time

Time ::= UTCTime

Content ::= OCTET STRING  --Left undefined in X.435 but found in X.411

ExternallyDefinedBodyPart ::= SEQUENCE  --See X.420, p.607
        {
        parameters      [0] ExternallyDefinedParameters OPTIONAL,
        data    ExternallyDefinedData
        }

ExternallyDefinedParameters ::= EXTERNAL

ExternallyDefinedData ::= EXTERNAL

ContentIntegrityCheck ::= SIGNATURE SEQUENCE--  --X.411, p.348
        {
        algorithm-identifier    ContentIntegrityAlgorithmIdentifier,
        content Content
        }

ContentIntegrityAlgorithmIdentifier ::= AlgorithmIdentifier

AlgorithmIdentifier ::= SEQUENCE  --X.509, p.79
        {
        algorithm       OBJECT IDENTIFIER,
        parameters      ANY DEFINED BY algorithm OPTIONAL
        }

SIGNATURE MACRO ::=  --X.509, Annex G
```

```
        BEGIN
        TYPE NOTATION ::= type(OfSignature)
        VALUE NOTATION ::= value(VALUE
                SEQUENCE
                        {
                        AlgorithmIdentifier,
                        ENCRYPTED OCTET STRING
                        }                       )
        END

--*** Begin ORName definition ***
ORName ::= [APPLICATION 0] SEQUENCE  --X.411, p.353
        {
        address COMPONENTS OF ORAddress,
        directory-name  [0] Name OPTIONAL
        }

ORAddress ::= SEQUENCE
        {
        standard-attributes     StandardAttributes,
        domain-defined-attributes       DomainDefinedAttributes OPTIONAL
        extension-attributes    ExtensionAttributes OPTIONAL
        }

StandardAttributes ::= SEQUENCE
         {
         country-name   CountryName OPTIONAL,
         administration-domain-name     AdministrationDomainName OPTIONAL,
         network-address        [0] NetworkAddress OPTIONAL,
         terminal-identifier    [1] TerminalIdentifier OPTIONAL,
         private-domain-name    [2] PrivateDomainName OPTIONAL,
         organization-name      [3] OrganizationName OPTIONAL,
         numeric-user-identifier        [4] NumericUserIdentifier OPTIONAL,
         personal-name  [5] PersonalName OPTIONAL,
         organizational-unit-names      [6] OrganizationalUnitNames OPTIONAL
         }

CountryName ::= [APPLICATION 1] CHOICE
         {
         x121-dcc-code  NumericString(SIZE(3)),  --ub-country-name-numeric-length
         iso-3166-alpha2-code   PrintableString(SIZE(2))  --ub-country-name-alpha-length
         }

AdministrationDomainName ::= [APPLICATION 2] CHOICE
         {
         numeric        NumericString(SIZE(0..16)),  --ub-domain-name-length
         printable      PrintableString(SIZE(0..16))  --ub-domain-name-length
         }

NetworkAddress ::= X121Address

X121Address ::= NumericString(SIZE(1..15))  --ub-x121-address-length

TerminalIdentifier ::= PrintableString(SIZE(1..24))  --ub-terminal-id-length

PrivateDomainName ::= CHOICE
        {
        numeric NumericString(SIZE(1..16)),  --ub-domain-name-length
        printable       PrintableString(SIZE(1..16))  --ub-domain-name-length
        }

OrganizationName ::= PrintableString(SIZE(1..64))  --ub-organization-name-length

NumericUserIdentifier ::= NumericString(SIZE(1..32))  --ub-numeric-user-id-length

PersonalName ::= SET
        {
        surname [0] PrintableString(SIZE(1..40)),  --ub-surname-length
        given-name      [1] PrintableString(SIZE(1..16)) OPTIONAL,  --ub-given-name-length
        initials        [2] PrintableString(SIZE(1..5)) OPTIONAL,  --ub-initials-length
        generation-qualifier    [3] PrintableString(SIZE(1..3)) OPTIONAL  --ub-generation-qualifier-length
        }

OrganizationalUnitNames ::= SEQUENCE SIZE(1..4) OF OrganizationUnitName  --ub-organizational-units

OrganizationUnitName ::= PrintableString(SIZE(1..32))  --ub-organizational-unit-name-length

DomainDefinedAttributes ::= SEQUENCE SIZE(1..4) OF DomainDefinedAttribute  --ub-domain-defined-attributes

DomainDefinedAttribute ::= SEQUENCE
        {
        type    PrintableString(SIZE(1..8)),  --ub-domain-defined-attribute-type-length
        value   PrintableString(SIZE(1..128))  --ub-domain-defined-attribute-value-length
        }

ExtensionAttributes ::= SET SIZE(1..256) OF ExtensionAttribute--  --ub-extension-attributes

ExtensionAttribute ::= SEQUENCE
        {
        extension-attribute-type        [0] EXTENSION-ATTRIBUTE,
        extension-attribute-value       [1] ANY DEFINED BY extension-attribute-type
        }

EXTENSION-ATTRIBUTE MACRO ::=
        BEGIN
        TYPE NOTATION ::= TYPE | empty
        VALUE NOTATION ::= value(VALUE INTEGER(0..256))--  --ub-extension-attributes
        END
```

```
Name ::= CHOICE  --X.501, p.42, only one possibility for now (Melbourne, 14-25 November 1988)
        {
        RDNSequence
        }

RDNSequence ::= SEQUENCE OF RelativeDistinguishedName

RelativeDistinguishedName ::= SET OF AttributeValueAssertion

AttributeValueAssertion ::= SEQUENCE
        {
        AttributeType,
        AttributeValue
        }

AttributeType ::= OBJECT IDENTIFIER

AttributeValue ::= ANY
--*** End ORName definition ***

--*** Begin OtherMessageDeliveryFields definition ***
OtherMessageDeliveryFields ::= SET  --X.411, p.338
        {
        content-type    DeliveredContentType,
        originator-name OriginatorName,
        original-encoded-information-types      [1] OriginalEncodedInformationTypes OPTIONAL,
        priority        Priority DEFAULT normal,
        delivery-flags  [2] DeliveryFlags OPTIONAL,
        other-recipient-names   [3] OtherRecipientNames OPTIONAL,
        this-recipient-name     [4] ThisRecipientName,
        originally-intended-recipient-name      [5] OriginallyIntendedRecipientName OPTIONAL,
        converted-encoded-information-types     [6] ConvertedEncodedInformationTypes OPTIONAL,
        message-submission-time [7] MessageSubmissionTime,
        content-identifier      [8] ContentIdentifier OPTIONAL
        extensions      [9] EXTENSIONS CHOSEN FROM  --See X.411, p.344 for EXTENSIONS macro definition
                {
              conversion-with-loss-prohibited,
                requested-delivery-method,
                physical-forwarding-prohibited,
                physical-forwarding-address-request,
                physical-delivery-mode,
                registered-mail-type,
                recipient-number-for-advice,
                physical-rendition-attributes,
                original-return-address,
                physical-delivery-report-request,
                original-certificate,
                message-token,
                content-confidentiality-algorithm-identifier
                content-integrity-check,
                message-origin-authentication-check,
                message-security-label,
                proof-of-delivery-request,
                redirection-history,
                dl-expansion-history
                } DEFAULT {}
        }

DeliveredContentType ::= CHOICE
        {
        built-in        [0] BuiltInContentType,
        external        ExternalContentType
        }

BuiltInContentType ::= [APPLICATION 6] INTEGER
        {
        unidentified (0),
        external (1),
        interpersonal-messaging-1984 (2),
        interpersonal-messaging-1988 (22)
        } (0..32767)  --ub-built-in-content-type

ExternalContentType ::= OBJECT IDENTIFIER

OriginatorName ::= OrAddressAndOrDirectoryName

OrAddressAndOrDirectoryName ::= ORName

OriginalEncodedInformationTypes ::= EncodedInformationTypes

EncodedInformationTypes ::= [APPLICATION 5] SET
        {
        built-in-encoded-information-types      [0] BuiltInEncodedInformationTypes,
        non-basic-parameters    COMPONENTS OF NonBasicParameters,
        external-encoded-information-types      [4] ExternalEncodedInformationTypes OPTIONAL
        }

BuiltInEncodedInformationTypes ::= BIT STRING
        {
        undefined (0),
        telex (1),
        ia5-text(2),
        g3-facsimile (3),
        g4-class-1 (4),
        teletex (5),
        videotex (6),
        voice (7),
        sfd (8),
        mixed-mode (9)
```

```
        } (SIZE(0..32))  --ub-built-in-encoded-information-types

NonBasicParameters ::= SET
        {
        g3-facsimile    [1] G3FacsimileNonBasicParameters DEFAULT {},
        teletex [2] TeletexNonBasicParameters DEFAULT {},
        g4-class-1-and-mixed-mode       [3] G4Class1AndMixedModeNonBasicParameters OPTIONAL
        }

G3FacsimileNonBasicParameters ::= BIT STRING
        {
        two-dimensional (8),
        fine-resolution (9),
        unlimited-length (20),
        b4-length (21),
        a3-width (22),
        b4-width (23),
        uncompressed (30)
        }

TeletexNonBasicParameters ::= SET
        {
        graphic-character-sets  [0] TeletexString OPTIONAL,
        control-character-sets  [1] TeletexString OPTIONAL,
        page-formats    [2] OCTET STRING OPTIONAL,
        miscellaneous-terminal-capabilities     [3] TeletexString OPTIONAL,
        private-use     [4] OCTET STRING OPTIONAL
                --maximum ub-teletex-private-use-length octets = 128
        }

G4Class1AndMixedModeNonBasicParameters ::= PresentationCapabilities

PresentationCapabilities ::= ANY

ExternalEncodedInformationTypes ::= SET SIZE(1..1024) OF ExternalEncodedInformationType  --ub-encoded-information-types

ExternalEncodedInformationType ::= OBJECT IDENTIFIER

Priority ::= [APPLICATION 7] ENUMERATED
        {
        normal (0),
        non-urgent (1),
        urgent (2)
        }

DeliveryFlags ::= BIT STRING
        {
        implicit-conversion-prohibited (1)  --prohibited=1, allowed=0
        } (SIZE(0..16))  --ub-bit-options

OtherRecipientNames ::= SEQUENCE SIZE(1..32767) OF OtherRecipientName  --ub-recipients

OtherRecipientName ::= ORAddressAndOrDirectoryName

ThisRecipientName ::= ORAddressAndOrDirectoryName

OriginallyIntendedRecipientName ::= ORAddressAndOrDirectoryName

ConvertedEncodedInformationTypes ::= EncodedInformationTypes

MessageSubmissionTime ::= Time

ContentIdentifier ::= [APPLICATION 10] PrintableString (SIZE(1..16))  --ub-content-id-length
--*** End OtherMessageDeliveryFields definition ***

--End Imported definitions (from X.411, X.420, X.509) **************************

--*******************************************************************************

END
```

Annexe C : PDU Pedi (Version "structurée")

```
--*****************************************************************************
--* CCITT X.435 Recommendation Pedi Protocol Data Unit                        *
--* Edited by Wei-Chao KAO              Nov. 1991, BIM                        *
--*                                                                           *
--* Cette version de PDU Pedi utilise un langage ASN.1 remanié pour mieux     *
--* mettre en évidence la structure générale du PDU Pedi                      *
--*****************************************************************************


        --NOTE :
        --Pour faciliter la lecture, nous écrivons
        --              xxx  [0] XXX OPTIONAL ::= { definition of XXX ..... }
        --au lieu de
        --              xxx  [0] XXX ::= { definition of XXX ..... } OPTIONAL


--Begin X.435 Pedi PDU ***************************************************************************************************************************************************
InformationObject ::= CHOICE
{
edim    [0] EDIM ::= SEQUENCE
        {
        heading Heading ::= SEQUENCE
                {
                this-EDIM       [1] ThisEDIMField ::= EDIMIdentifier ::= SET
                        {
                        user    [0] ORName,  --Refer to the end of this text
                        user-relative-identifier        [1] LocalReference ::=
                                PrintableString(SIZE(0..ub-local-reference))  --64
                        },
                originator      [2] OriginatorField OPTIONAL ::= ORName,  --Refer to the end of this text
                recipients      [3] RecipientsField OPTIONAL ::= SET OF
                        RecipientsSubField ::= SEQUENCE
                                {
                                recipient       [1] RecipientField ::= ORName,  --Refer to the end of this text
                                action-request  [2] ActionRequestField DEFAULT {id-for-action} ::= OBJECT IDENTIFIER,  --0
                                EDI-notification-requests-field [3] EDINotificationRequestsField OPTIONAL ::= SEQUENCE
                                        {
                                        edi-notification-requests       [0] EDINotificationRequests DEFAULT {} ::=
                                                BIT STRING
                                                {
                                                pn (0),
                                                nn (1),
                                                fn (2)
                                                } (SIZE(0..ub-bit-options)),  --16
                                        edi-notification-security       [1] EDINotificationSecurity DEFAULT {} ::=
                                                BIT STRING
                                                {
                                                proof (0),
                                                non-repudiation (1)
                                                } (SIZE(0..ub-bit-options)),  --16
                                        edi-reception-security  [2] EDIReceptionSecurity DEFAULT {} ::=
                                                BIT STRING
                                                {
                                                proof (0),
                                                non-repudiation (1)
                                                } (SIZE(0..ub-bit-options))  --16
                                        },
                                responsability-passing-allowed  [4] ResponsabilityPassingAllowedField DEFAULT FALSE ::= BOOLEAN,
```

```
                --Begin Fields from EDIFACT UNB
                interchange-recipient   [5] InterchangeRecipientField OPTIONAL ::= SEQUENCE
                        {
                        recipient-identification-code   [0] IdentificationCode ::=
                                T61String(SIZE(1..ub-identification-code)),  --35
                        identification-code-qualifier   [1] IdentificationCodeQualifier OPTIONAL ::=
                                T61String(SIZE(1..ub-identification-code-qualifier)),  --4
                        routing-address [2] RoutingAddress OPTIONAL ::=
                                T61String(SIZE(1..ub-routing-address))  --14
                        },
                recipient-reference     [6] RecipientReferenceField OPTIONAL ::= SEQUENCE
                        {
                        recipient-reference     [0] RecipientReference ::=
                                T61String(SIZE(1..ub-recipient-reference)),  --14
                        recipient-reference-qualifier   [1] RecipientReferenceQualifier OPTIONAL ::=
                                T61String(SIZE(1..ub-recipient-reference-qualifier))  --2
                        },
                interchange-control-reference   [7] InterchangeControlReferenceFiled OPTIONAL ::=
                        T61String(SIZE(1..ub-interchange-control-reference)),  --14
                processing-priority-code        [8] ProcessingPriorityCodeField OPTIONAL ::=
                        T61String(SIZE(1..ub-processing-priority-code)),  --1
                acknowledgement-request [9] AcknowledgementRequestField DEFAULT FALSE ::= BOOLEAN
                communications-agreement-id     [10] CommunicationsAgreementIdField OPTIONAL ::=
                        T61String(SIZE(1..ub-communications-agreement-id)),  --35
                test-indicator  [11] TestIndicatorField DEFAULT FALSE ::= BOOLEAN,
                --End Fileds from EDIFACT UNB

                --Begin Fields from ANSIX12 ISA
                authorization-information       [12] AuthorizationInformationField OPTIONAL ::= SEQUENCE
                        {
                         authorization-information      [0] AuthorizationInformation ::=
                                T61String (SIZE(1..ub-authorization-information)),  --10
                        authorization-information-qualifier     [1] AuthorizationInformationQualifier OPTIONAL ::=
                                T61String (SIZE(1..ub-authorization-information-qualifier)),  --2
                        },
                --End Fields from ANSIX12 ISA

                recipient-extensions    [13] RecipientExtensionsField OPTIONAL := SET OF
                        RecipientExtensionSubField ::=
                                ExtensionField ::= SEQUENCE
                                        {
                                        type    [0] EDIM-EXTENSION MACRO ::=
                                                BEGIN
                                                TYPE NOTATION ::= Data Type Critical | empty
                                                VALUE NOTATION ::= value(VALUE OBJECT IDENTIFIER)
                                                Data Type ::= type (X) Default | empty
                                                Default ::= ''DEFAULT'' value (X) | empty
                                                Critical ::= ''CRITICAL'' | empty
                                                END,
                                        criticality     [1] Criticality DEFAULT FALSE ::= BOOLEAN,
                                        value   [2] ANY DEFINED BY type DEFAULT NULL NULL
                                        }
                },
edin-receiver   [4] EDINReceiverField OPTIONAL ::= SEQUENCE
        {
        edin-receiver-name      [0] ORName,  --Refer to the end of this text
        original-edim-identifier        [1] EDIMIdentifier OPTIONAL ::= SET
                {
                user    [0] ORName,  --Refer to the end of this text
                user-relative-identifier        [1] LocalReference ::=
                        PrintableString(SIZE(0..ub-local-reference))  --64
                },
        first-recipient [2] FirstRecipientField OPTIONAL ::=
                ORName  --Refer to the end of this text
```

```
        },
responsability-forwarded        [5] ResponsabilityForwarded DEFAULT FALSE ::= BOOLEAN,
edi-bodypart-type       [6] EDIBodypartType DEFAULT {id-edifact-ISO646} ::= OBJECT IDENTIFIER,  --0
incomplete-copy [7] IncompleteCopyField DEFAULT FALSE ::= BOOLEAN,
expiry-time     [8] ExpiryTimeField OPTIONAL ::= UTCTime,
related-messages        [9] RelatedMessagesField OPTIONAL ::= SEQUENCE OF
        RelatedMessageReference ::= CHOICE
                {
                edi-message-reference   [0] EDIMIdentifier ::= SET
                        {
                        user    [0] ORName,  --Refer to the end of this text
                        user-relative-identifier        [1] LocalReference ::=
                                PrintableString(SIZE(0..ub-local-reference))  --64
                        },
                external-message-reference      [1] ExternalMessageReference ::= EXTERNAL
                },
obsoleted-EDIMs [10] ObsoletedEDIMsField OPTIONAL ::= SEQUENCE OF
        ObsoletedEDIMsSubfiled ::=
                EDIMIdentifier ::= SET
                        {
                        user    [0] ORName,  --Refer to the end of this text
                        user-relative-identifier        [1] LocalReference ::=
                                PrintableString(SIZE(0..ub-local-reference))  --64
                        },
edi-application-security-elements       [11] EDIApplicationSecurityElementsField OPTIONAL ::= SEQUENCE
        {
        edi-application-security-element        [0] EDIApplicationSecurityElement OPTIONAL ::=
                BIT STRING(SIZE(0..ub-edi-application-security-elements)),  --8191
        edi-encrypted-primary-bodypart [1] BOOLEAN OPTIONAL,
        edi-application-security-extensions     [2] EDIApplicationSecurityExtensions OPTIONAL ::= SEQUENCE OF
                EDIApplicationSecurityExtension ::= ExtensionField ::= SEQUENCE         --''ExtensionsField'' in X.435, p.104
                                                                                --a mistake from CCITT??
                        {
                        type    [0] EDIM-EXTENSION MACRO ::=
                                BEGIN
                                TYPE NOTATION ::= Data Type Critical | empty
                                VALUE NOTATION ::= value(VALUE OBJECT IDENTIFIER)
                                Data Type ::= type (X) Default | empty
                                Default ::= ''DEFAULT'' value (X) | empty
                                Critical ::= ''CRITICAL'' | empty
                                END,
                        criticality     [1] Criticality DEFAULT FALSE ::= BOOLEAN,
                        value   [2] ANY DEFINED BY type DEFAULT NULL NULL
                        }
        },
cross-referencing-information   [12] CrossReferencingInformationField OPTIONAL ::= SET OF
        {
        application-cross-reference     [0] ApplicationCrossReference ::= OCTET STRING,
        message-reference       [1] MessageReference OPTIONAL ::=
                EDIM Identifier ::= SET
                        {
                        user    [0] ORName,  --Refer to the end of this text
                        user-relative-identifier        [1] LocalReference ::=
                                PrintableString(SIZE(0..ub-local-reference))  --64
                        },
        body-part-reference     [2] BodyPartReference ::= INTEGER
        },

--Begin Fields from EDIFACT Interchange
edi-message-type        [13] EDIMessageTypeField OPTIONAL ::= SET OF
        EDIMessageTypeFieldSubField ::=
                T61String(SIZE(1..ub-edi-message-type)),  --6
service-string-advice   [14] ServiceStringAdviceField OPTIONAL ::= SEQUENCE
        {
```

```
            component-data-element-separator        [0] ComponentDataElementSeparator ::=
                    OCTET STRING(SIZE(1)),
            data-element-separator  [1] DataElementSeparator ::=
                    OCTET STRING(SIZE(1)),
            decimal-notation        [2] DecimalNotation ::=
                    OCTET STRING(SIZE(1)),
            release-indicator       [3] ReleaseIndicator OPTIONAL ::=
                    OCTET STRING(SIZE(1)),
            reserved        [4] Reserved OPTIONAL ::=
                    OCTET STRING(SIZE(1)),
            segment-terminator      [5] SegmentTerminator ::=
                    OCTET STRING(SIZE(1))
            },
    syntax-identifier       [15] SyntaxIdentifierField OPTIONAL ::= SEQUENCE
            {
            syntaxe-identifier      SyntaxeIdentifier ::=
                    T61String(SIZE(1..ub-syntax-identifier)),  --4
            syntaxe-version SyntaxeVersion ::=
                    PrintableString (SIZE(1..ub-syntax-version))  --5
            },
    interchange-sender      [16] InterchangeSenderField OPTIONAL ::= SEQUENCE
            {
            sender-identification   [0] IdentificationCode ::=
                    T61String(SIZE(1..ub-identification-code)),  --35
            identification-code-qualifier   [1] IdentificationCodeQualifier OPTIONAL ::=
                    T61String(SIZE(1..ub-identification-code-qualifier)),  --4
            address-for-reverse-routing     [2] RoutingAddress OPTIONAL ::=
                    T61String(SIZE(1..ub-routing-address))  --14
            }
    date-and-time-of-preparation    [17] DateAndTimeOfPreparationField OPTIONAL ::=
            UTCTime,
    application-reference   [18] ApplicationReferenceField OPTIONAL ::=
            T61String(SIZE(1..ub-application-reference))  --14
    --End Fields fram EDIFACT
    },  --End of Heading

body    Body ::= SEQUENCE
    {
    primary-body-part       PrimaryBodyPart ::= CHOICE
            {
            edi-body-part   [0] EDIBodyPart ::= OCTET STRING,
            forwarded-EDIM  [1] EDIMBodyPart ::= SEQUENCE
                    {
                    parameters      [0] MessageParameters OPTIONAL ::= SET
                            {
                            delivery-time   [0] MessageDeliveryTime OPTIONAL ::= Time ::=  --See X.411
                                    UTCTime,
                            delivery-envelope       [1] OtherMessageDeliveryFields OPTIONAL ::= SET  --See X.411
                                    {
                                    content-type   DeliveredContentType ::= CHOICE
                                            {
                                            built-in        BuiltInContentType ::= [Application 6] Integer
                                                    {
                                                    unidentified (0),
                                                    external (1),
                                                    interpersonal-messaging-1984 (2),
                                                    interpersonal-messaging-1988 (22)
                                                    } (0..ub-built-in-content-type),  --32767
                                            external        ExternalContentType ::= OBJECT IDENTIFIER
                                            },
                                    originator-name OriginatorName ::= OrAddressAndOrDirectoryName ::=
                                            ORName,  --Refer to the end of this text
                                    original-encoded-information-types      [1] OriginalEncodedInformationTypes OPTIONAL ::=
                                            EncodedInformationTypes ::= [Application 5] SET
```

```
                {
                built-in-encoded-information-types      [0] BuiltInEncodedInformationTypes ::= BIT STRING
                        {
                        undefined (0),
                        telex (1),
                        ia5-text(2),
                        g3-facsimile (3),
                        g4-class-1 (4),
                        teletex (5),
                        videotex (6),
                        voice (7),
                        sfd (8),
                        mixed-mode (9)
                        } (SIZE(0..ub-built-in-encoded-information-types)),  --32
                non-basic-parameters    COMPONENTS OF NonBasicParameters ::= SET
                        {
                        g3-facsimile    [1] G3FacsimileNonBasicParameters DEFAULT {} ::= BIT STRING
                                {
                                 two-dimensional (8),
                                 fine-resolution (9),
                                 unlimited-length (20),
                                 b4-length (21),
                                 a3-width (22),
                                 b4-width (23),
                                 uncompressed (30)
                                 },
                        teletex [2] TeletexNonBasicParameters DEFAULT {} ::= SET
                                {
                                graphic-character-sets  [0] TeletexString OPTIONAL,
                                control-character-sets  [1] TeletexString OPTIONAL,
                                page-formats    [2] OCTET STRING OPTIONAL,
                                miscellaneous-terminal-capabilities     [3] TeletexString OPTIONAL,
                                private-use     [4] OCTET STRING OPTIONAL
                                        --maximum ub-teletex-private-use-length octets = 128
                                },
                        g4-class-1-and-mixed-mode       [3] G4Class1AndMixedModeNonBasicParameters OPTIONAL ::=
                                PresentationCapabilities ::= ANY
                        },
                external-encoded-information-types      [4] ExternalEncodedInformationTypes OPTIONAL ::=
                        SET SIZE(1..ub-encoded-information-types) OF  --1024
                                ExternalEncodedInformationType ::= OBJECT IDENTIFIER
                },
priority        Priority DEFAULT normal ::= [Application 7] ENUMERATED
        {
        normal (0),
        non-urgent (1),
        urgent (2)
        },
delivery-flags  [2] DeliveryFlags OPTIONAL ::= BIT STRING
        {
        implicit-conversion-prohibited (1),  --prohibited=1, allowed=0
        } SIZE(0..ub-bit-options),  --16
other-recipient-names   [3] OtherRecipientNames OPTIONAL ::= SEQUENCE SIZE(1..ub-recipients) OF  --32767
        OtherRecipientName ::= OrAddressAndOrDirectoryName ::=
                ORName,  --Refer to the end of this text
this-recipient-name     [4] ThisRecipientName ::=
        OrAddressAndOrDirectoryName ::=
                ORName,  --Refer to the end of this text
originally-intended-recipient-name      [5] OriginallyIntendedRecipientName OPTIONAL ::=
        OrAddressAndOrDirectoryName ::=
                ORName,  --Refer to the end of this text
converted-encoded-information-types     [6] ConvertedEncodedInformationTypes OPTIONAL ::=
        EncodedInformationTypes ::= [Application 5] SET
                {
```

```
                built-in-encoded-information-types      [0] BuiltInEncodedInformationTypes ::= BIT STRING
                        {
                        undefined (0),
                        telex (1),
                        ia5-text(2),
                        g3-facsimile (3),
                        g4-class-1 (4),
                        teletex (5),
                        videotex (6),
                        voice (7),
                        sfd (8),
                        mixed-mode (9)
                        } (SIZE(0..ub-built-in-encoded-information-types)),  --32
                non-basic-parameters    COMPONENTS OF NonBasicParameters ::= SET
                        {
                        g3-facsimile    [1] G3FacsimileNonBasicParameters DEFAULT {} ::= BIT STRING
                                {
                                 two-dimensional (8),
                                 fine-resolution (9),
                                 unlimited-length (20),
                                 b4-length (21),
                                 a3-width (22),
                                 b4-width (23),
                                 uncompressed (30)
                                 },
                        teletex [2] TeletexNonBasicParameters DEFAULT {} ::= SET
                                {
                                graphic-character-sets  [0] TeletexString OPTIONAL,
                                control-character-sets  [1] TeletexString OPTIONAL,
                                page-formats    [2] OCTET STRING OPTIONAL,
                                miscellaneous-terminal-capabilities     [3] TeletexString OPTIONAL,
                                private-use     [4] OCTET STRING OPTIONAL
                                        --maximum ub-teletex-private-use-length octets = 128
                                },
                        g4-class-1-and-mixed-mode       [3] G4Class1AndMixedModeNonBasicParameters OPTIONAL ::=
                                PresentationCapabilities ::= ANY
                        },
                external-encoded-information-types      [4] ExternalEncodedInformationTypes OPTIONAL ::=
                        SET SIZE(1..ub-encoded-information-types) OF  --1024
                                ExternalEncodedInformationType ::= OBJECT IDENTIFIER
                },
message-submission-time [7] MessageSubmissionTime ::= Time ::= UTCTime,
content-identifier      [8] ContentIdentifier OPTIONAL ::=
        [Application 10] PrintableString (SIZE(1..ub-content-id-length)),  --16
extensions      [9] EXTENSIONS CHOSEN FROM  --See X.411, p.344 for EXTENSIONS macro definition
        {
         conversion-with-loss-prohibited,
        requested-delivery-method,
        physical-forwarding-prohibited,
        physical-forwarding-address-request,
        physical-delivery-mode,
        registered-mail-type,
        recipient-number-for-advice,
        physical-rendition-attributes,
        original-return-address,
        physical-delivery-report-request,
        original-certificate,
        message-token,
        content-confidentiality-algorithm-identifier,
        content-integrity-check,
        message-origin-authentication-check,
        message-security-label,
        proof-of-delivery-request,
        redirection-history,
```

```
                                        dl-expansion-history
                                        } DEFAULT {}
                                },
                        other-parameters        [2] EDISupplementaryInformation OPTIONAL ::=
                                IA5String(SIZE(1..ub-supplementary-info-length))  --256
                        },
                data    [1] MessageData ::= SEQUENCE
                        {
                        heading Heading,  --Refer to the Heading definition at the beginning of this text
                        body    BodyOrRemoved ::= SEQUENCE
                                {
                                primary-or-removed      PrimaryOrRemoved ::= CHOICE
                                        {
                                        removed-edi-body        [0] NULL,
                                        primary-body-part       [1] EXPLICIT PrimaryBodyPart
                                                --Recursive definition: see above for PrimaryBodyPart definition
                                        },
                                additional-body-parts   AdditionalBodyParts OPTIONAL ::= SEQUENCE OF CHOICE
                                        {
                                        external-body-part      [0] EDIM-externallyDefinedBodyPart ::= SEQUENCE
                                                {
                                                body-part-reference     [0] BodyPartReference OPTIONAL ::= INTEGER  --unique within an EDIM
                                                external-body-part      [1] ExternallyDefinedBodyPart ::= SEQUENCE  --See X.420, p.607
                                                        {
                                                        parameters      [0] ExternallyDefinedParameters OPTIONAL ::= EXTERNAL,
                                                        data    ExternallyDefinedData ::= EXTERNAL
                                                        }
                                                },
                                        place-holder    [1] BodyPartPlaceHolder ::= EDIM-ExternallyDefinedBodyPart ::= SEQUENCE
                                                {
                                                body-part-reference     [0] BodyPartReference OPTIONAL ::= INTEGER  --unique within an EDIM
                                                external-body-part      [1] ExternallyDefinedBodyPart ::= SEQUENCE  --See X.420, p.607
                                                        {
                                                        parameters      [0] ExternallyDefinedParameters OPTIONAL ::= EXTERNAL,
                                                        data    ExternallyDefinedData ::= EXTERNAL
                                                        }
                                                }
                                        }
                                }
                        }
                },
        additional-body-parts   OtherBodyParts OPTIONAL ::= SEQUENCE OF EDIM-ExternallyDefinedBodyPart ::= SEQUENCE
                {
                body-part-reference     [0] BodyPartReference OPTIONAL ::= INTEGER  --unique within an EDIM
                external-body-part      [1] ExternallyDefinedBodyPart ::= SEQUENCE  --See X.420, p.607
                        {
                        parameters      [0] ExternallyDefinedParameters OPTIONAL ::= EXTERNAL,
                        data    ExternallyDefinedData ::= EXTERNAL
                        }
                }
        }  --End of Body
},  --End of EDIM


edin    [1] EDIN ::= CHOICE
        {
        positive-notification   [0] PositiveNotificationFields ::= SEQUENCE
                {
                pn-common-fields        [0] CommonFields,  --Refer to the end of this text
                pn-supplementary-information    [1] EDISupplementaryInformation OPTIONAL ::= IA5String(SIZE(1..ub-supplementary-info-length)),  --256
                pn-extensions   [2] PNExtensionsField OPTIONAL ::= SET OF
                        PNExtensionSubField ::= ExtensionField ::= SEQUENCE
                        {
```

```
                        type    [0] EDIM-EXTENSION MACRO ::=
                                BEGIN
                                TYPE NOTATION ::= Data Type Critical | empty
                                VALUE NOTATION ::= value(VALUE OBJECT IDENTIFIER)
                                Data Type ::= type (X) Default | empty
                                Default ::= ''DEFAULT'' value (X) | empty
                                Critical ::= ''CRITICAL'' | empty
                                END,
                        criticality     [1] Criticality DEFAULT FALSE ::= BOOLEAN,
                        value   [2] ANY DEFINED BY type DEFAULT NULL NULL
                        }
        },
negative-notification   [1] NegativeNotificationFields ::= SEQUENCE
        {
        nn-common-fields        [0] CommonFields,  --Refer to the end of this text
        nn-reason-code  [1]NNReasonCodeField ::= CHOICE
                {
                nn-ua-ms-reason-code    [0] NNUAMSReasonCodeField ::= SEQUENCE
                        {
                        nn-ua-ms-basic-code     [0] NNUAMSBasicCodeField ::= INTEGER
                                {
                                unspecified (0),
                                cannot-deliver-to-user (1),
                                delivery-timeout (2),
                                message-discarded (3),
                                subscription-terminated (4),
                                forwarding-error (5),
                                security-error (6)
                                } (1..ub-reason-code),  --32767
                        nn-ua-ms-diagnostic     [1] NNUAMSDiagnosticField OPTIONAL ::= INTEGER
                                {
                                --General diagnostic codes
                                protocol-violation (1),
                                edim-originator-unknown (2),
                                edim-recipient-unknown (3),
                                edim-recipient-ambiguous (4),
                                action-request-not-supported (5),
                                edim-expired (6),
                                edim-obsoleted (7),
                                duplicate-edim (8),
                                unsupported-extension (9),
                                incomplete-copy-rejected (10),
                                edim-too-large-for-application (11),
                                forwarded-edim-not-delivered (12),
                                forwarded-edim-delivered-time-out (13),
                                forwarded-loop-detected (14),
                                unable-to-accept-responsability (15),
                                --Interchange header diagnostic codes
                                interchange-sender-unknown (16),
                                interchange-recipient-unknown (17),
                                invalid-heading-field (18),
                                invalid-bodypart-type (19),
                                invalid-message-type (20),
                                invalid-syntax-id (21),
                                --Security error diagnostic codes
                                message-integrity-failure (22),
                                forwarded-message-integrity-failure (23),
                                unsupported-algorithm (24),
                                decryption-failed (25),
                                token-error (26),
                                unable-to-sign-notification (27),
                                unable-to-sign-message-receipt (28),
                                authentication-failure (29),
                                security-context-failure (30),
```

```
                        message-sequence-failure (31),
                        message-security-labelling-failure (32),
                        repudiation-failure (33),
                        proof-of-failure (34)
                        } (1..ub-reason-code)  --32767
                },
        nn-user-reason-code     [1] NNUserReasonCodeField ::= SEQUENCE
                {
                nn-user-basic-code      [0] NNUserBasicCodeField ::= INTEGER
                        {
                        unspecified (0),
                        syntax-error (1),
                        interchange-sender-unknown (2),
                        interchange-recipient-unknown (3),
                        invalid-heading-field (4),
                        invalid-body-part-type (5),
                        invalid-message-type (6),
                        functional-groups-not-supported (7),
                        subscription-terminated (8),
                        no-bilateral-agreement (9),
                        user-defined-reason (10)
                        } (1..ub-reason-code),  --32767
                nn-user-diagnostic      [1] NNUserDiagnosticField OPTIONAL ::= INTEGER
                },
        nn-pdau-reason-code     [2] NNPDAUReasonCodeField ::= SEQUENCE
                {
                nn-pdau-basic-code      [0] NNPDAUBasicCodeField ::= INTEGER
                        {
                        unspecified (0),
                        undeliverable-mail (1),
                        physical-rendition-not-performed (2)
                        } (1..ub-reason-code),  --32767
                nn-pdau-diagnostic      [1] NNPDAUDiagnosticField OPTIONAL ::= INTEGER
                        {
                        undeliverable-mail-physical-delivery-address-incorrect (1),
                        undeliverable-mail-physical-delivery-office-incorrect-or-invalid (2),
                        undeliverable-mail-physical-delivery-address-incomplete (3),
                        undeliverable-mail-recipient-unknown (4),
                        undeliverable-mail-recipient-deceased (5),
                        undeliverable-mail-recipient-refused-to-accept (6),
                        undeliverable-mail-organization-expired (7),
                        undeliverable-mail-recipient-did-not-claim (8),
                        undeliverable-mail-recipient-changed-address-permanently (9),
                        undeliverable-mail-recipient-changed-address-temporarily (10),
                        undeliverable-mail-recipient-changed-temporary-address (11),
                        undeliverable-mail-new-address-unknown (12),
                        undeliverable-mail-recipient-did-not-want-forwarding (13),
                        undeliverable-mail-originator-prohibited-forwarding (14),
                        physical-rendition-attributes-not-supported (15)
                        } (1..ub-reason-code)  --32767
                }
        },
nn-supplementary-information    [2] EDISupplementaryInformation OPTIONAL ::= IA5String(SIZE(1..ub-supplementary-info-length)), --256
nn-extensions   [3] NNExtensionsField OPTIONAL ::= SET OF NNExtensionsSubField ::= ExtensionField ::= SEQUENCE
        {
        type    [0] EDIM-EXTENSION MACRO ::=
                BEGIN
                TYPE NOTATION ::= Data Type Critical | empty
                VALUE NOTATION ::= value(VALUE OBJECT IDENTIFIER)
                Data Type ::= type (X) Default | empty
                Default ::= ''DEFAULT'' value (X) | empty
                Critical ::= ''CRITICAL'' | empty
                END,
        criticality     [1] Criticality DEFAULT FALSE ::= BOOLEAN,
```

```
                value   [2] ANY DEFINED BY type DEFAULT NULL NULL
                }
        },
forwarded-notification  [2] ForwardedNotificationFields ::= SEQUENCE
        {
        fn-common-fields        [0] CommonFields,  --Refer to the end of this text
        forwarded-to    [1] ForwardedTo ::= ORName,  --Refer to the end of this text
        fn-reason-code  [2] FNReasonCodeField ::= CHOICE
                {
                fn-ua-ms-reason-code    [0] FNUAMSReasonCodeField ::= SEQUENCE
                        {
                        fn-ua-ms-basic-code     [0] FNUAMSBasicCodeField ::= INTEGER
                                {
                                unspecified (0),
                                onward-routing (1),
                                recipient-unknown (2),
                                originator-unknown (3),
                                forwarded-by-edi-ms (4)
                                } (0..ub-reason-code),  --32767
                        fn-ua-ms-diagnostic     [1] FNUAMSDiagnosticField OPTIONAL ::= INTEGER
                                {
                                recipient-name-changed (0),
                                recipient-name-deleted (1)
                                } (0..ub-reason-code),  --32767
                        fn-security-check       [2] FNUAMSSecurityCheckField DEFAULT FALSE ::= BOOLEAN
                        },
                fn-user-reason-code     [1] FNUserReasonCodeField ::= SEQUENCE
                        {
                        fn-user-basic-code      [0] FNUserBasicCodeField ::= INTEGER
                                {
                                unspecified (0),
                                forwarded-for-archiving (1),
                                forwarded-for-information (2),
                                forwarded-for-additional-action (3),
                                subscription-changed (4),
                                heading-field-not-supported (5),
                                bodypart-type-not-supported (6),
                                message-type-not-supported (7),
                                syntax-identifier-not-supported (8),
                                interchange-sender-unknown (9),
                                interchange-recipient-unknown (10),
                                user-defined-reason (11)
                                } (0..ub-reason-code),  --32767
                        fn-user-diagnostic      [1] FNUserDiagnosticField OPTIONAL ::= INTEGER
                        },
                fn-pdau-reason-code     [2] FNPDAUReasonCodeField ::= SEQUENCE
                        {
                        fn-pdau-basic-code      [0] FNPDAUBasicCodeField ::= INTEGER
                                {
                                unspecified (0),
                                forwarded-for-physical-rendition-and-delivery (1)
                                } (0..ub-reason-code),  --32767
                        fn-pdau-diagnostic      [1] FNPDAUDiagnosticField OPTIONAL ::= INTEGER
                        }
                }
        fn-supplementary-information    [3] EDISupplementaryInformation OPTIONAL ::= IA5String(SIZE(1..ub-supplementary-info-length)), --256
        fn-extensions   [4] FNExtensionsField ::= SET OF FNExtensionsSubField ::= ExtensionField ::= SEQUENCE
                {
                type    [0] EDIM-EXTENSION MACRO ::=
                        BEGIN
                        TYPE NOTATION ::= Data Type Critical | empty
                        VALUE NOTATION ::= value(VALUE OBJECT IDENTIFIER)
                        Data Type ::= type (X) Default | empty
                        Default ::= ''DEFAULT'' value (X) | empty
```

```
                        Critical ::= ''CRITICAL'' | empty
                        END,
                    criticality     [1] Criticality DEFAULT FALSE ::= BOOLEAN,
                    value   [2] ANY DEFINED BY type DEFAULT NULL NULL
                    }
            }
        }  --End of EDIN
}  --End of InformationObject


--Begin CommonFields definition ************************************************
CommonFields ::= SEQUENCE
        {
        subject-edim    [1] SubjectEDIMField ::= EDIMIdentifier ::= SET
                {
                user    [0] ORName,  --Refer to the end of this text
                user-relative-identifier        [1] LocalReference ::=
                        PrintableString(SIZE(0..ub-local-reference))  --64
                },
        edin-originator [2] EDINOriginatorField ::= ORName,  --Refer to the end of this text
        first-recipient [3] FirstRecipientField ::= ORName,  --Refer to the end of this text
        notification-time       [4] NotificationTimeField OPTIONAL ::= UTCTime,
        notification-security-elements  [5] SecurityElementsField OPTIONAL ::= SEQUENCE
                {
                original-content        [0] Content OPTIONAL ::= OCTET STRING,  --Undefined in X.435 but found in X.411
                original-content-integrity-check        [1] ContentIntegrityCheck OPTIONAL ::= SIGNATURE SEQUENCE
                        --See X.411 p.348 for ContentIntegrityCheck, X.509 Annex H for SIGNATURE macro definition
                        {
                        algorithm-identifier    ContentIntegrityAlgorithmIdentifier ::= AlgorithmIdentifier ::= SEQUENCE
                                {
                                algorithm       OBJECT IDENTIFIER,
                                parameters      ANY DEFINED BY algorithm OPTIONAL
                                },
                        content Content ::= OCTET STRING,  --Undefined in X.435 but found in X.411
                        },
                edi-application-security-elements       [2] EDIApplicationSecurityElementsField OPTIONAL ::= SEQUENCE
                        {
                        edi-application-security-element        [0] EDIApplicationSecurityElement OPTIONAL ::=
                                BIT STRING(SIZE(0..ub-edi-application-security-elements)),  --8191
                        edi-encrypted-primary-bodypart  [1] BOOLEAN OPTIONAL,
                        edi-application-security-extensions     [2] EDIApplicationSecurityExtensions OPTIONAL ::= SEQUENCE OF
                                EDIApplicationSecurityExtension ::= ExtensionField ::= SEQUENCE         --''ExtensionsField'' in X.435, p.104
                                                                                        --a mistake from CCITT??
                                        {
                                        type    [0] EDIM-EXTENSION MACRO ::=
                                                BEGIN
                                                TYPE NOTATION ::= Data Type Critical | empty
                                                VALUE NOTATION ::= value(VALUE OBJECT IDENTIFIER)
                                                Data Type ::= type (X) Default | empty
                                                Default ::= ''DEFAULT'' value (X) | empty
                                                Critical ::= ''CRITICAL'' | empty
                                                END,
                                        criticality     [1] Criticality DEFAULT FALSE ::= BOOLEAN,
                                        value   [2] ANY DEFINED BY type DEFAULT NULL NULL
                                        }
                        },
                security-extensions     [3] SecurityExtensionsField OPTIONAL ::=SET OF
                        SecurityExtensionSubField ::= ExtensionField ::= SEQUENCE
                                {
                                type    [0] EDIM-EXTENSION MACRO ::=
                                        BEGIN
                                        TYPE NOTATION ::= Data Type Critical | empty
                                        VALUE NOTATION ::= value(VALUE OBJECT IDENTIFIER)
```

```
                                Data Type ::= type (X) Default | empty
                                Default ::= ''DEFAULT'' value (X) | empty
                                Critical ::= ''CRITICAL'' | empty
                                END,
                        criticality     [1] Criticality DEFAULT FALSE ::= BOOLEAN,
                        value   [2] ANY DEFINED BY type DEFAULT NULL NULL
                        }
                },
        edin-initiator  [6] EDINInitiatorField ::= ENUMERATED
                {
                internal-ua (0),
                external-ua (1),
                internal-ms (2)
                }
        notifications-extensions        [7] NotificationExtensionsField OPTIONAL ::= SET OF
                NotificationExtensionsSubField ::= ExtensionField ::= SEQUENCE
                        {
                        type    [0] EDIM-EXTENSION MACRO ::=
                                BEGIN
                                TYPE NOTATION ::= Data Type Critical | empty
                                VALUE NOTATION ::= value(VALUE OBJECT IDENTIFIER)
                                Data Type ::= type (X) Default | empty
                                Default ::= ''DEFAULT'' value (X) | empty
                                Critical ::= ''CRITICAL'' | empty
                                END,
                        criticality     [1] Criticality DEFAULT FALSE ::= BOOLEAN,
                        value   [2] ANY DEFINED BY type DEFAULT NULL NULL
                        }
        }
--End CommonFields definition ***************************************************


--Begin ORName definition *********************************************************
ORName ::= [APPLICATION 0] SEQUENCE  --X.411, p.353
        {
        address COMPONENTS OF ORAddress ::= SEQUENCE
                {
                standard-attributes     StandardAttributes ::= SEQUENCE
                        {
                        country-name    CountryName OPTIONAL ::= [APPLICATION 1] CHOICE
                                {
                                x121-dcc-code   NumericString(SIZE(ub-country-name-numeric-length)),  --3
                                iso-3166-alpha2-code    PrintableString(SIZE(ub-country-name-alpha-length))  --2
                                },
                        administration-domain-name      AdministrationDomainName OPTIONAL ::= [APPLICATION 2] CHOICE
                                {
                                numeric NumericString(SIZE(0..ub-domain-name-length)),  --16
                                printable       PrintableString(SIZE(0..ub-domain-name-length))  --16
                                },
                        network-address [0] NetworkAddress OPTIONAL ::= x121Address ::= NumericString(SIZE(1..ub-x121-address-length)),  --15
                        terminal-identifier     [1] TerminalIdentifier OPTIONAL ::= PrintableString(SIZE(1..ub-terminal-id-length)),  --24
                        private-domain-name     [2] PrivateDomainName OPTIONAL ::= CHOICE
                                {
                                numeric NumericString(SIZE(0..ub-domain-name-length)),  --16
                                printable       PrintableString(SIZE(0..ub-domain-name-length))  --16
                                },
                        organization-name       [3] OrganizationName OPTIONAL ::= PrintableString(SIZE(1..ub-organization-name-length)),  --64
                        numeric-user-identifier [4] NumericUserIdentifier OPTIONAL ::= NumericString(SIZE(1..ub-numeric-user-id-length)),  --32
                        personal-name   [5] PersonalName OPTIONAL ::= SET
                                {
                                surname [0] PrintableString(SIZE(1..ub-surname-length)),  --40
                                given-name      [1] PrintableString(SIZE(1..ub-given-name-length)) OPTIONAL,  --16
                                initials        [2] PrintableString(SIZE(1..ub-initials-length)) OPTIONAL,  --5
                                generation-qualifier    [3] PrintableString(SIZE(1..ub-generation-qualifier-length)) OPTIONAL  --3
```

```
                        },
                organizational-unit-names       [6] OrganizationalUnitNames OPTIONAL ::= SEQUENCE SIZE(1..ub-organizational-units) OF  --4
                        OrganizationUnitName ::= PrintableString(SIZE(1..ub-organizational-unit-name-length))  --32
                },
        domain-defined-attributes       DomainDefinedAttributes OPTIONAL SEQUENCE SIZE(1..ub-domain-defined-attributes) OF
                DomainDefinedAttribute ::= SEQUENCE  --4
                {
                type    PrintableString(SIZE(1..ub-domain-defined-attribute-type-length)),  --8
                value   PrintableString(SIZE(1..ub-domain-defined-attribute-value-length))  --128
                },
        extension-attributes    ExtensionAttributes OPTIONAL ::= SET SIZE(1..ub-extension-attributes) OF  --256
                ExtensionAttribute ::= SEQUENCE
                {
                extension-attribute-type        [0] EXTENSION-ATTRIBUTE MACRO ::=
                        BEGIN
                        TYPE NOTATION ::= TYPE | empty
                        VALUE NOTATION ::= value(VALUE INTEGER(0..ub-extension-attributes))  --256
                        END,
                extension-attribute-value       [1] ANY DEFINED BY extension-attribute-type
                }
        },
    directory-name  [0] Name OPTIONAL ::= CHOICE  --X.501, p.42, only one possibility for now (Melbourne, 14-25 November 1988)
        {
        RDNSequence ::= SEQUENCE OF
                RelativeDistinguishedName ::= SET OF
                        AttributeValueAssertion ::= SEQUENCE
                                {
                                AttributeType ::= OBJECT IDENTIFIER,
                                AttributeValue ::= ANY
                                }
        }
    }
--End ORName definition ************************************************

--End X.435 Pedi PDU ************************************************************************************************************************************************
```